UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

INFERENCIA DE MÁQUINAS DE ESTADO DESDE BIG DATA PARA EL ETIQUETADO ESTÁTICO Y CARACTERIZACIÓN DE CONDUCTAS DINÁMICAS: GENERACIÓN DE LAS QUERIES DE DOMINIO

Jorge Guillén Alonso
Tutor: Alejandro Echeverría Rey
Ponente: Alfonso Ortega de La Puente

Junio 2017

INFERENCIA DE MÁQUINAS DE ESTADO DESDE BIG DATA PARA EL ETIQUETADO ESTÁTICO Y CARACTERIZACIÓN DE CONDUCTAS DINÁMICAS: GENERACIÓN DE LAS QUERIES DE DOMINIO

AUTOR: Jorge Guillén Alonso
TUTOR: Alejandro Echeverría Rey

Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2017

Resumen (castellano)

En la actualidad, la cantidad de información que un usuario genera es enorme, tanto la información que él mismo genera de manera consciente, como la que se deduce de sus hábitos en la red o incluso de su residencia. Es posible cotejar la información interna de que dispone una empresa, la información tradicional, con información más dinámica generada por redes sociales u otros sistemas de reciente implantación. Mientras que la información tradicional generalmente está compuesta de estructuras estáticas según el tipo de cliente, la información dinámica está formada por datos no estructurados, que a priori no se pueden utilizar ni almacenar de manera eficiente, como es el caso de un servicio de ayuda online personalizado.

A estos datos internos de la empresa se les pueden añadir datos externos que ayuden a caracterizar al cliente, como los datos relativos a su navegación en internet o datos sociodemográficos. Al agregar estos datos se puede crear lo que se conoce como el ciclo de vida de un cliente, esto es algo muy demandado por las empresas, ya que supone un gran potencial de negocio puesto que el mayor conocimiento de los clientes permite tanto la detección de fugas, como la creación de ofertas más personalizadas y efectivas para los clientes.

Para llevar a cabo la manipulación de estos volúmenes de datos es necesario un tipo de tecnología capaz de manipular datos no estructurados y de realizar consultas bajo demanda. Actualmente es un proceso muy complejo y costoso, en el que es difícil integrar información abundante y ágil. Apache Spark junto con el sistema de archivos distribuidos de Hadoop están especialmente diseñados para cumplir con estos requisitos, por lo que serán fundamentales para el éxito de este Trabajo.

Otra parte fundamental es el algoritmo de clasificación de los clientes. Para este propósito se ha elegido realizar la clasificación con un algoritmo de inferencia de máquinas de estado. Esto permite realizar diagramas con los estados de los clientes, y de este modo poder predecir cuáles podrían ser sus siguientes acciones.

Este Trabajo Fin de Grado se realiza con la colaboración de la empresa consultora Gain Dynamics y tiene como objetivo poner en marcha su proyecto de confección de ciclo de vida del cliente mediante un nuevo enfoque utilizando su infraestructura de servidores para instalar en ella la última versión de las tecnologías Big Data y crear las queries de dominio que generen una tabla preparada para ser introducida en un algoritmo de inferencia de máquinas de estado.

Además, este Trabajo Fin de Grado se relacionará con el Trabajo de Fin de Grado “Inferencia de máquinas de estado desde Big Data para el etiquetado estático y caracterización de conductas dinámicas: generador del modelo”, que creará el modelo de la máquina de estados. Este Trabajo forma parte de un proyecto más amplio, y es una línea de investigación que Gain Dynamics persigue con el objetivo de crear un ciclo de vida para clientes.

Abstract (English)

At present, the amount of information that a user generates is enormous, the information that he generates consciously, the one that is deduced from his habits on the web or even his residence. It is possible to compare internal information available to the company, traditional information, with more dynamic information generated by social networks or other systems of recent implementation. While traditional information is composed of static structures according to the type of client, the dynamic information is formed by unstructured data, which cannot be used efficiently, as in the case of a personalized online help service.

This internal data of the company can be added to data that help characterize the client, such as data related to their Internet browsing or socio-demographic data. By adding this data you can create what is known as the life cycle of a customer, this is something very demanded by the companies, since it supposes a great potential of business since the greater knowledge of the clients allow the detection of leaks or create more personalized and effective offers for customers.

To manipulate these volumes of data is required a type of technology capable of manipulating unstructured data and performing on-demand queries. It is currently a very complex and expensive process, in which it is difficult to integrate abundant and agile information. Apache Spark along with Hadoop's distributed file system is specially designed to meet these requirements, so they are critical to the success of this work.

Another fundamental part is the algorithm of classification of the clients. For this purpose it has been chosen to perform the classification with an inference algorithm of state machines. This allows to make diagrams with the states of the clients, and thus to be able to predict which are the following actions.

This End-of-Grade Work is carried out with the collaboration of the consultant Gain Dynamics and aims to launch its client life cycle design project through a new approach in its server infrastructure to install the latest version of Technologies Big Data and create the domain questions that generate a table prepared to be entered into a state machine inference algorithm.

In addition, this End-of-Grade Work was related to the End-of-Grade work " Inferencia de máquinas de estado desde Big Data para el etiquetado estático y caracterización de conductas dinámicas: generador del modelo", which create the model of the machine of states. This work is part of a larger project, and is a line of research that Gain Dynamics pursues with the aim of creating a life cycle for customers.

Palabras clave (castellano)

Big Data, Apache Hadoop, Apache Spark, Apache Ambari, HDFS, Hive, Apache Zeppelin, CentOS, Máquinas de estado, Tabla transactional, Tabla contextual, Scala.

Keywords (inglés)

Big Data, Apache Hadoop, Apache Spark, Apache Ambari, HDFS, Hive, Apache Zeppelin, CentOS, State machines, Transactional table, Contextual table. Scala.

Agradecimientos

Quiero expresar mi gratitud a todas las personas que me han ayudado en la realización de este Trabajo Fin de Grado. En particular, quiero dar las gracias a Alejandro Echeverría por su seguimiento de mi trabajo diario y sus inestimables comentarios, y a Alfonso Ortega por su valiosa ayuda. Asimismo quiero expresar mi agradecimiento a los compañeros de Gain Dynamics que me aportaron sus críticas y sugerencias.

También quiero expresar mi gratitud a Gain Dynamics por poner sus servidores a mi disposición, sin cuya aportación la realización de este Trabajo no hubiera sido posible.

Por último, quiero dar las gracias a los miembros de mi familia que me obsequiaron con su comprensión y apoyo durante mis estudios.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

1 Introducción

1.1 Motivación

Vivimos en el mundo de la información, se calcula que actualmente la información almacenada en la web está entorno a 5k petabyte, repartidos entre decenas de miles de millones de URLs y se prevé un crecimiento x50 entre 2010 y 2020. El 70% de esta información está creada por los usuarios finales, el 39% de la población humana utiliza internet, más del 97% de la comunicación mundial es vía internet y entre el 10-15% del comercio mundial es electrónico [1]. La red, y los datos que contiene son cada vez más importantes, hemos pasado de limitarnos a escribir correos instantáneos a escribir blogs, expresar y compartir nuestras vivencias por las redes sociales, dedicar nuestro ocio al visionado de series y películas, escuchar música, etc. Toda esta información es un reflejo bastante fiel de la persona que la ha creado, y lo más importante, es información accesible que la persona transmite al momento, de manera natural y sin ser percibido como un esfuerzo extra. Permite conocer su estado de ánimo, sus gustos, datos personales, etc.

Esta cantidad de información personal es la que promovió la creación de este proyecto. Si existiese una forma de poder clasificar a los clientes y predecir sus movimientos supondría un potencial de negocio enorme para las empresas [2]. Actualmente existen formas de enriquecer los datos del cliente para lograr cierta clasificación, una muy usada es establecer dónde vive ese cliente y caracterizarlo en función de su zona. Pero aunque dos clientes vivan en el mismo lugar pueden ser radicalmente opuestos. Con las nuevas tecnologías se puede intentar humanizar a ese cliente, obtener un perfil más representativo de él, se puede investigar sus redes sociales, con Data Management Platform (DMP) se pueden ver sus gustos de navegación, utilizar datos sociodemográficos para establecer un cierto nivel económico, etc. Si se juntan todos estos datos con los datos internos de la empresa interesada, se podría crear un mapa de clientes con el cual poder establecer dónde se encuentra y a dónde es probable que vaya un cliente en particular.

1.2 Objetivos

El objetivo principal de este TFG es el desarrollo parcial de un sistema que generalice la inferencia de máquinas de estado (o variantes suyas) para que, a partir de grandes cantidades de información (Big Data), se automatice en la medida de lo posible tanto el etiquetado estático como la caracterización de las conductas dinámicas del dominio. Una de las componentes identificadas en el sistema tiene que ver con la generación automática de las consultas que hay que realizar al repositorio de datos del dominio y la preparación de éste.

Este TFG se enmarca en ese contexto, en conseguir una base sólida para un proyecto de clasificación de clientes con información de dimensión estática (caracterización de la estructura de las entidades involucradas) y dinámica (caracterización respecto a su evolución en el tiempo).

Esta base consta de:

- Sistema operativo: Un sistema operativo estable diseñado para su uso en servidores y preparado para una carga de trabajo constante.
- Entorno de desarrollo: Un entorno capaz de manipular grandes volúmenes de datos no estructurados de manera rápida.
- Lenguaje de desarrollo: Un lenguaje capaz de realizar consultas en tiempo real.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Capítulo 1: Introducción**
  En este apartado se explicarán la motivación que llevó a la creación de este Trabajo, los objetivos que abarca y la estructura de la memoria.

- **Capítulo 2: Estado del arte**
  Se definirá de forma compacta qué es Big Data y sus herramientas, y qué es una máquina de estados y sus posibilidades en la clasificación.

- **Capítulo 3: Diseño**
  Se tratará de ofrecer una vista global de los requisitos necesarios de software y hardware, y de la arquitectura de Hadoop que se va a implementar y por qué.

- **Capítulo 4: Desarrollo**
  En este capítulo se expondrá el proceso llevado a cabo para la consecución del objetivo, desde la instalación hasta la creación de la tabla para la máquina de estados, y las dificultades y retos encontrados.

- **Capítulo 5: Integración, pruebas y resultados**
  En este apartado se aportarán pruebas del correcto funcionamiento del servidor y las queries.

- **Capítulo 6: Conclusiones y trabajo futuro**
  Se expondrán las conclusiones obtenidas con el Trabajo Fin de Grado y se comentarán los siguientes pasos a seguir.

2 Estado del arte

2.1 Introducción

Como se ha mencionado en el apartado *1.2 Objetivos*, para la realización de este proyecto es necesario un tipo de tecnología específicamente diseñada para trabajar con grandes volúmenes de datos y con datos no estructurados.

Este proyecto se creó porque actualmente no existe ninguna librería para Big Data que se encargue de realizar el preprocesamiento de datos para máquinas de estados, y el correcto formato de los datos que alimentan a las máquinas de estados es necesario para la creación del ciclo de vida del cliente.

Aunque el estudio de la forma en la que se tratarán los datos finales y cuál sería más óptima para nuestro caso no está en el alcance de este proyecto, si es necesario un conocimiento medio de qué es una máquina de estados y qué tipo de datos necesita.

En los apartados siguientes se detalla el tipo de tecnología necesaria para el éxito del trabajo y los conocimientos necesarios sobre las máquinas de estados para la realización del repositorio de datos de dominio.

2.2 Big Data

Big Data es necesario para este proyecto porque se basa en procesamiento y almacenamiento de grandes volúmenes de datos de manera rápida y eficaz. Este proyecto manejará una gran cantidad de datos provenientes de las conductas de los clientes y necesitará de procesos rápidos para actualizar los datos de las máquinas de estados.

2.2.1 Apache Hadoop

Apache Hadoop es un proyecto de desarrollo de software de código abierto para la computación distribuida, con persistencia ante fallos y fácilmente escalable gracias a su arquitectura Master-Slave. Además es ampliamente utilizado por grandes empresas como IBM, Banco Santander, Ebay, Facebook, etc [3].

La biblioteca de software Apache Hadoop es un framework que permite el procesamiento distribuido de grandes conjuntos de datos a través de clusters de servidores usando modelos de programación sencillos. Está diseñado para escalar de servidores individuales a miles de máquinas, cada una ofreciendo almacenamiento y computación local. En lugar de confiar en hardware para ofrecer alta disponibilidad, la propia biblioteca de Hadoop está diseñada para detectar y manejar fallos en la capa de aplicación, por lo que ofrece un servicio altamente disponible encima de un grupo de equipos, cada uno de los cuales puede ser propenso a fallos[4].

El ecosistema Hadoop es muy diverso y crece cada día, actualmente los módulos principales de Hadoop son:

- **Hadoop common:** Utilidades que soportan los otros módulos de Hadoop.
- **Hadoop Distributed File System (HDFS):** Un sistema de archivos distribuido que proporciona acceso de alto rendimiento a los datos de la aplicación.
- **Hadoop YARN:** Un framework para programación de tareas y gestión de recursos.
- **Hadoop MapReduce:** Un sistema basado en YARN para el procesamiento de grandes conjuntos de datos.

Hadoop y todos sus componentes son open-source.

2.2.2 Bases de Datos

Hadoop cuenta con varias formas de almacenar datos:

- **Hadoop Distributed File System (HDFS):** Como ya se explicó en el apartado *2.2.1 Apache Hadoop,* HDFS está centrado en ser un sistema distribuido que proporciona acceso de alto rendimiento. Está optimizado para trabajar con hardware de bajo costo y pese a tener muchas similitudes con otros sistemas de archivos distribuidos HDFS es altamente tolerante a fallos.
- **Apache Hive:** Es un data warehouse construido sobre Hadoop que facilita la lectura, escritura y gestión de grandes conjuntos de datos que residen en el almacenamiento distribuido mediante HQL, un lenguaje SQL-like que es traducido a trabajos MapReduce. En Hadoop solamente se ejecutan procesos batch.
- **Apache HBase:** Es una Base de Datos NoSQL que se ejecuta sobre HDFS. A diferencia de Hive, las operaciones en HBase se ejecutan en tiempo real. HBase se divide en tablas y las tablas se dividen en familias de columnas que agrupan un número de éstas.

2.2.3 Apache Spark

Spark es un motor de cálculo rápido y general para los datos de Hadoop. Proporciona una interfaz de programación de aplicaciones centradas en una estructura de datos denominada resilient distributed dataset (RDD), un multiset de sólo lectura de datos distribuidos en un cluster de máquinas tolerante a fallos [5].

Spark proporciona un modelo de programación que soporta una amplia gama de aplicaciones, incluyendo ETL, aprendizaje automático, procesamiento de flujos y computación gráfica.

2.2.4 Lenguajes de programación

Spark Core API soporta una amplia variedad de lenguajes, desde SQL hasta gráficos. De todos los disponibles, Scala es el lenguaje más interesante, es un lenguaje sofisticado con una sintaxis flexible, y se ha demostrado que en carga es más rápido que Python o Java.

Al trabajar con muchos núcleos, en general el rendimiento no es un factor importante en la elección del lenguaje de programación. Sin embargo, cuando hay lógica de procesamiento significativa y se debe tener en cuenta el rendimiento o el número de núcleos no es elevado, Scala ofrece un mejor rendimiento [6].

2.3 Máquinas de estados

El motivo del tratamiento de los datos en este proyecto para que los utilice una máquina de estados es porque permite reflejar fácil e intuitivamente el ciclo de vida de un cliente, ver el estado actual del cliente y ver sus posibles estados siguientes, aspecto vital del proyecto en el que se engloba este Trabajo de Fin de Grado.

2.3.1 Descripción

Se denomina máquina de estados a un modelo de comportamiento de un sistema con entradas y salidas, en donde las salidas dependen no sólo de las señales de entradas actuales sino también de las anteriores. Las máquinas de estado se definen como un conjunto de estados que sirve de intermediario entre la entrada y la salida, haciendo que las señales de entrada determinen el estado actual de la máquina, de tal forma que la salida dependa del estado actual y las entradas [7].

El único tipo de máquina de estados que se puede modelar en un ordenador es el que se denomina máquina de estados finitos (FSM) que, como su nombre indica, se compone de un conjunto finito de estados. Este término se utiliza como sinónimo de máquina de estados, puesto que es el único tipo que se puede crear.

2.3.2 Máquinas de estados orientadas a la clasificación

Una máquina de estado es un modelo de comportamiento de un sistema con entradas y salidas, en donde la salida no sólo depende de las señales de entrada actuales, sino que también depende de manera indirecta de las señales anteriores, por medio de los estados intermedios que llevaron al estado actual [8].

Una máquina de estados está compuesta por un conjunto de estados que sirve de intermediario entre entradas y salidas, haciendo que el historial de entradas determine los posibles estados actuales de la máquina.

Para la realización una máquina de estados orientada al ciclo de vida del cliente, modelo que se implementará en futuros trabajos al estar fuera del alcance de este Trabajo Fin de Grado, será necesaria la colaboración de un experto analista de la empresa cliente para la creación del esquema orientado al ciclo de vida del cliente de la máquina de estados.

También se puede utilizar un dominio genérico para la creación del esquema y orientar la máquina de estados al ciclo de un agente.

La máquina de estado se alimentará de dos tablas, la tabla transaccional y la contextual.

- **La tabla transaccional:** constará de las variables temporales (variables de orden y tiempo transcurrido entre estados) y de las variables de los propios estados, variables tanto de la información interna del cliente como de la navegación en internet de éste. Estos datos se actualizarán periódicamente.

- **La tabla contextual:** contendrá información relativa a valores sociodemográficos, valores que por su naturaleza varían poco, y por consiguiente se actualizarán con menor frecuencia que la tabla transaccional.

3 Diseño

3.1 Objetivos de diseño

El objetivo de este apartado es establecer los requisitos necesarios para la realización de este trabajo, las necesidades a nivel software, el entorno en el que se va a desarrollar el proyecto, el Sistema Operativo óptimo para el rendimiento y que acepte este entorno de desarrollo y las necesidades hardware que se necesitan para ejecutar esta tecnología.

También se explicará la arquitectura de Hadoop y se detallarán los módulos principales que se van a instalar.

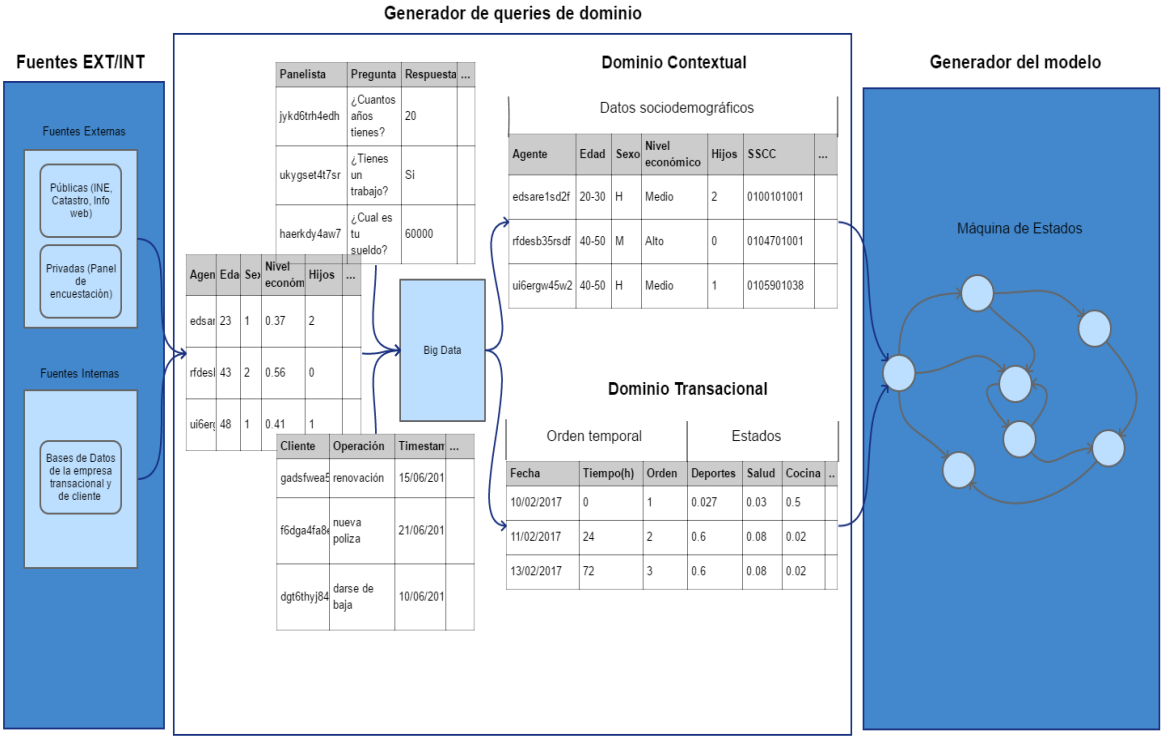


Figura 3-1: Arquitectura del proyecto

En la figura 3-1 se puede observar el flujo de datos del proyecto que delimita este Trabajo de Fin de Grado. Desde la entrada de la información y como está estructurada, hasta las dos tablas resultado del procesamiento de Big Data y cómo estas tablas crean la máquina de estados y la alimentan.

3.2 Requisitos de diseño

A continuación se detallan los requerimientos software y hardware para la correcta consecución del proyecto.

3.2.1 Software

- **Plataforma Hadoop**

Como se detalló en el apartado *1.2 Objetivos*, es necesario un tipo de tecnología que pueda tratar grandes volúmenes de datos de manera rápida y eficiente. Apache Hadoop no sólo nos proporciona esto si no que nos da muchas herramientas para manejar este tipo de problemas. La plataforma de Hadoop permite acciones como monitorizar los servidores, dividir las tareas, realizar mantenimientos o gestionar balanceos de carga. Hadoop es el que se encarga de todos estos factores para que el programador sólo tenga que centrarse en qué quiere hacer y no en cómo optimizar los recursos.

Además, si fuera necesario, con Hadoop se puede ampliar el cluster de manera sencilla debido a que Hadoop emplea la arquitectura Master-Slave, por lo que resulta fácil aumentar el número de servidores, en comparación con otros tipos de gestión de recursos. Su distribución típica para clusters de tamaño medio y alto, es tener entorno a dos servidores masters, dependiendo del nivel carga, y el resto de servidores como slaves, por lo que mientras se tengan los servidores masters necesarios, añadir servidores slaves será rápido.

- **Sistema Operativo**

Como se ha explicado, este proyecto requiere del framework de Apache Hadoop. Por lo tanto es requisito indispensable que el Sistema Operativo sea capaz de manejar e instalar esta tecnología.

3.2.2 Hardware

Este proyecto está pensado para ser llevado a cabo con el uso de las tecnologías de Big Data que nos brinda Apache. Actualmente el volumen de carga con el que se trabaja en Gain Dynamics no es muy alto, pero sí lo será en un futuro, por lo que teniendo en cuenta lo descrito en el apartado *2.2.2 Bases de Datos* sobre las distintas formas de almacenar datos, y aunque Hadoop puede trabajar con equipos básicos, si se necesita alto rendimiento, es recomendable que se utilicen como servidores, máquinas diseñadas para este propósito, con discos duros persistentes ante fallos y componentes que permitan soportar altos niveles de estrés y carga continuada, como por ejemplo, los procesadores Intel Xeon.

En la empresa Gain Dynamics se dispone de dos servidores específicamente comprados pensando en las tareas relacionadas con Big Data y sus necesidades de potencia y espacio. Y de ser necesario, podría fácilmente expandirse gracias a la arquitectura Master-Slave de la que dispone Hadoop.

Tener los nodos en local en vez de en la nube permite un mayor y más estable flujo de datos, ya que no depende de internet. Por este motivo se ha optado por el ámbito local para los servidores.

3.3 Arquitectura de Hadoop

En este apartado se va a describir cuáles son los módulos fundamentales que se van a instalar para este proyecto y por qué son necesarios.

Aunque Hadoop puede instalarse con múltiples gestores, los módulos principales siempre son instalados en todas las versiones e instalaciones posibles, puesto que son la piedra angular de este sistema. Por lo tanto YARN y HDFS siempre se instalan.

Para este proyecto se ha decidido realizar una instalación manual, lo cual implica utilizar los medios que proporciona Apache para su instalación y no utilizar ningún tipo de gestor externo. Esto permite tener un control más personal de todos los módulos instalados y de sus versiones, como contrapartida, el proceso de instalación se hace más complejo y el mantenimiento de los servidores está a cargo de la empresa propietaria.

En la figura 3.1 se muestra un diseño esquemático de los componentes básicos y más comunes de los que suelen componerse toda instalación de Apache Hadoop. Sobre este esquema pueden añadirse múltiples módulos especializados para satisfacer ciertas necesidades específicas. En el caso concreto de este proyecto, no ha sido necesaria la incorporación de más módulos que los que proporciona el gestor web de Hadoop, Apache Ambari.

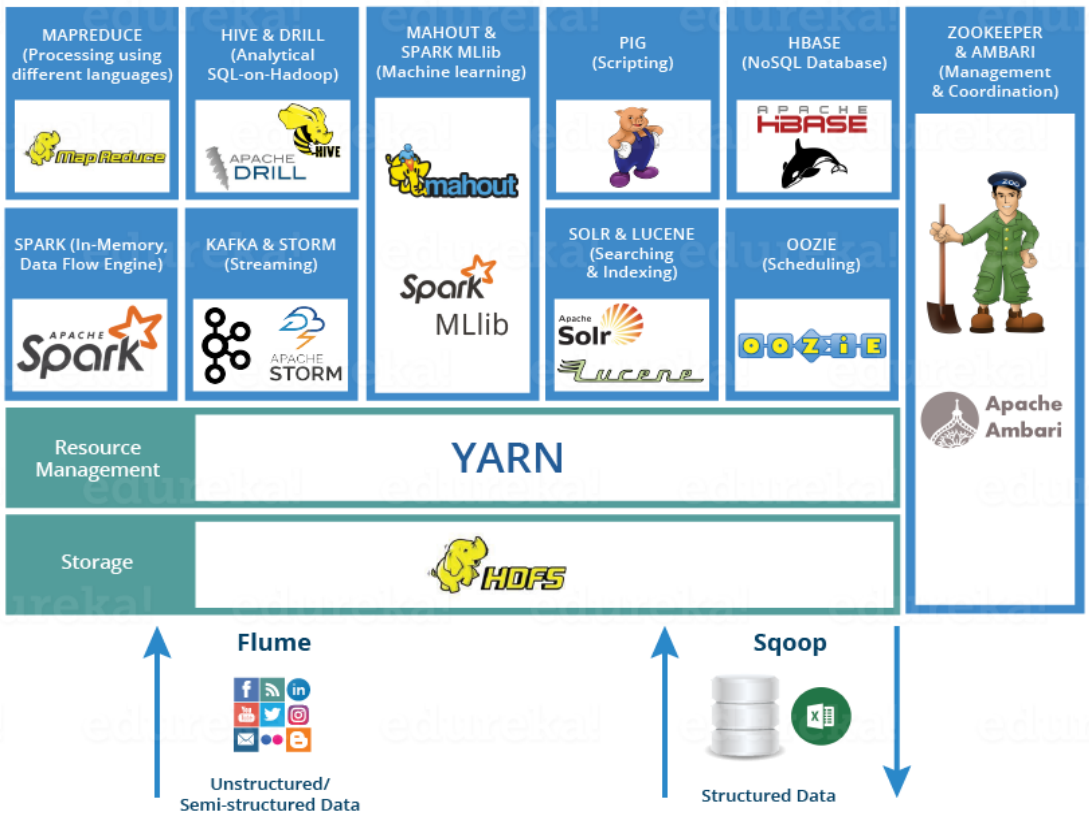


Figura 3-2: Arquitectura Hadoop

En este Trabajo de Fin de Grado se han instalado y utilizado los módulos de Apache que aparecen en la figura 3-2.

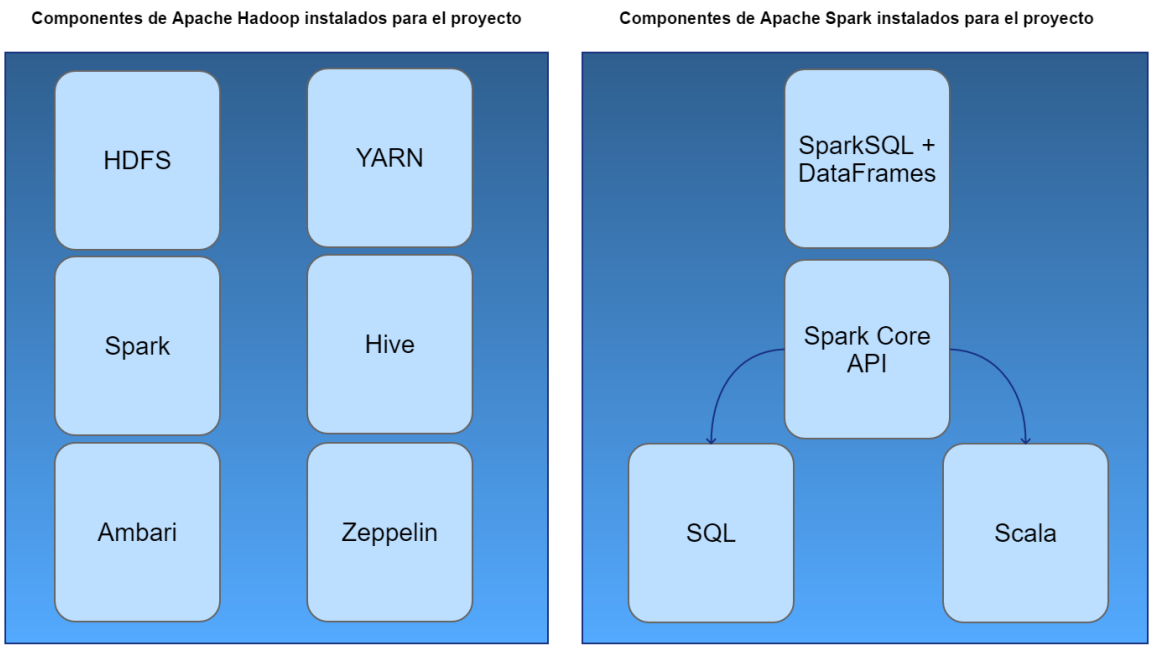


Figura 3-3: Componentes del proyecto

Las razones por las que se ha instalado cada módulo son:

- HDFS como sistema de archivos distribuido.
- YARN como control y MapReduce.
- Ambari para la instalación y gestión de recursos.
- Hive para el almacenamiento final de las tablas.
- Spark y sus API para el procesamiento.
- Zeppelin para ejecutar sentencias Spark y SQL.

La empresa Gain Dynamics ya disponía de servidores con la tecnología de Apache Hadoop instalada, pero debido a la velocidad del desarrollo de esta tecnología, la antigüedad de los servicios que tenían instalados impedía la realización de este proyecto.

Por este motivo se decidió realizar una nueva instalación con la última versión disponible de Apache Hadoop y personalizar la instalación de los módulos de acuerdo a las necesidades del proyecto, tanto el que abarca este Trabajo de Fin de Grado, como el proyecto global de la empresa.

También se ha realizado la instalación de otros módulos mostrados en la figura 3-4, que se utilizarán en el desarrollo del proyecto global.

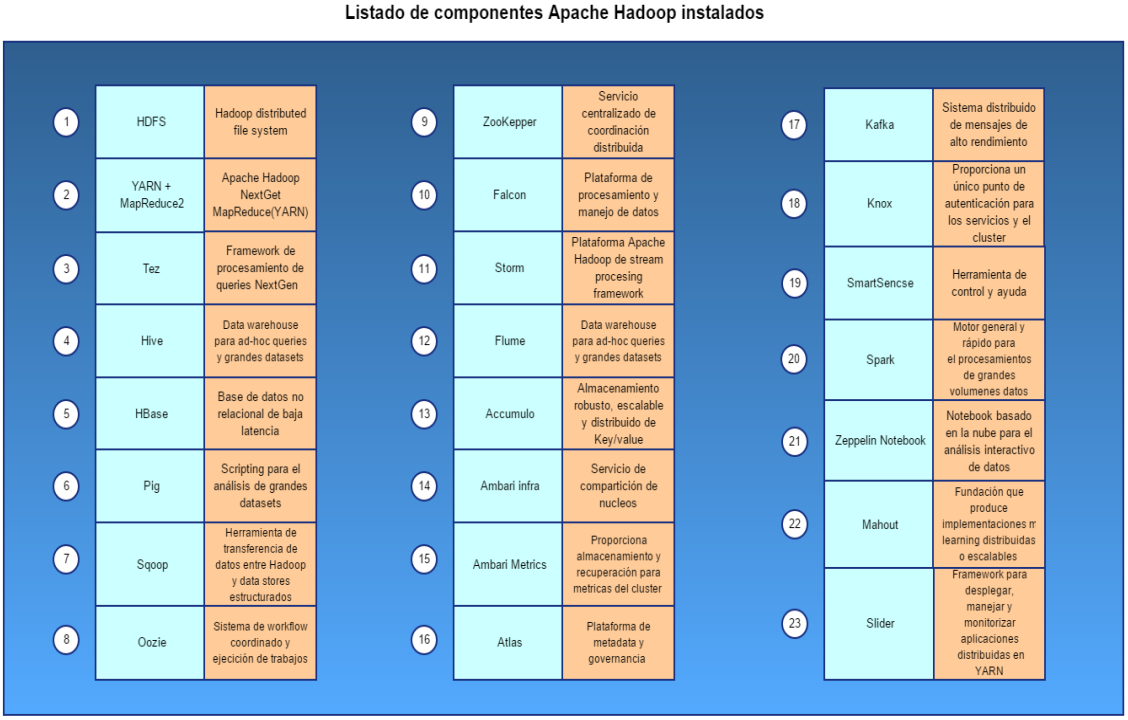


Figura 3-4: Componentes del proyecto global

3.3.1 Módulos básicos: HDFS y YARN

A lo largo de esta memoria se ha introducido y explicado HDFS, el sistema de archivos distribuidos de Hadoop. Es una pieza clave para el funcionamiento de Hadoop y permite almacenar archivos estructurados y no estructurados de manera distribuida, y su instalación no sólo es altamente recomendable, sino que es obligatoria.

Del módulo YARN en cambio se ha hablado poco en esta memoria para la enorme importancia que tiene y al igual que en el caso de HDFS es un módulo indispensable. YARN, "Yet Another Resource Negociator" u "otro negociador de recursos" por sus siglas en inglés, es el encargado de realizar el balanceo de carga en el conjunto de recursos. Su principal característica es permitir a Hadoop separar las capacidades de gestión de recursos y planificación de MapReduce del componente de procesamiento de datos.

La idea es tener un global ResourceManager (RM) y por aplicación un ApplicationMaster (AM). El framework de cálculo de datos está formado por el ResourceManager, que arbitra los recursos entre todas las aplicaciones del sistema, y el NodeManager, que es un framework por máquina encargado de monitorizar y reportar el uso de recursos al ResourceManager.

ApplicationMaster es una librería de framework específica y tiene la tarea de negociar los recursos desde el ResourceManager y trabajar con el/los NodeManager para ejecutar y monitorizar tareas.

ResourceManager tiene dos componentes principales:

- Scheduler encargado de asignar recursos a diversas aplicaciones de acuerdo a determinadas restricciones. No realiza la monitorización.

- ApplicationsManager encargado de aceptar trabajos, negociar el contenedor para el ApplicationMaster y proporcionar un servicio de reinicio en caso de fallo del contenedor.

3.3.2 Spark

Como ya se detalló en el apartado *2.2.3 Apache Spark*, Spark es un motor de cálculo rápido y general para los datos de Hadoop. El motivo de su integración en este proyecto es simple, es el motor que se utilizará para la manipulación de los datos, su almacenamiento y futuro procesamiento.

Spark proporcionará la capacidad de realizar operaciones en tiempo real, de manera que los datos de nueva inclusión afecten lo antes posible a las máquinas de estado y éstas muestren el siguiente paso del cliente.

En la figura 3-5 se puede apreciar el ecosistema que conforma Spark, desde las librerías como MLlib hasta Spark Core API, el encargado de los lenguajes de programación que soporta Spark.

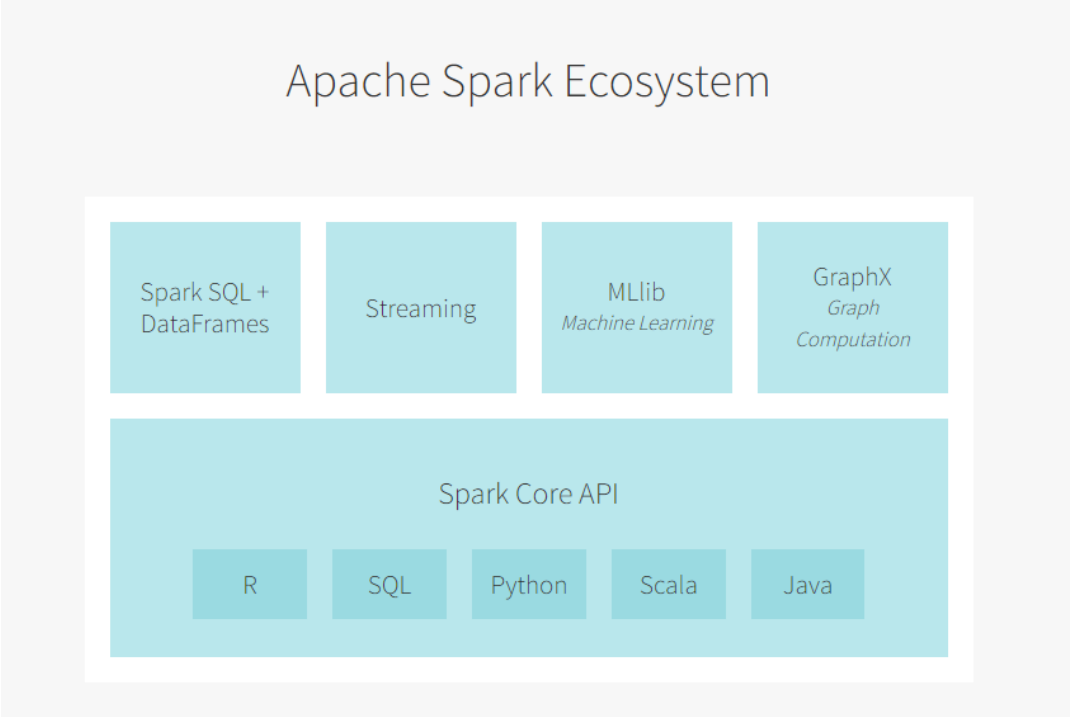


Figura 3-5: Ecosistema Spark

Spark será la herramienta que nos permita leer los archivos de HDFS y procesarlos para convertirlos en las tablas transaccional y contextual que en el futuro utilizará la máquina de estados. Como aparece en la figura 3-6.

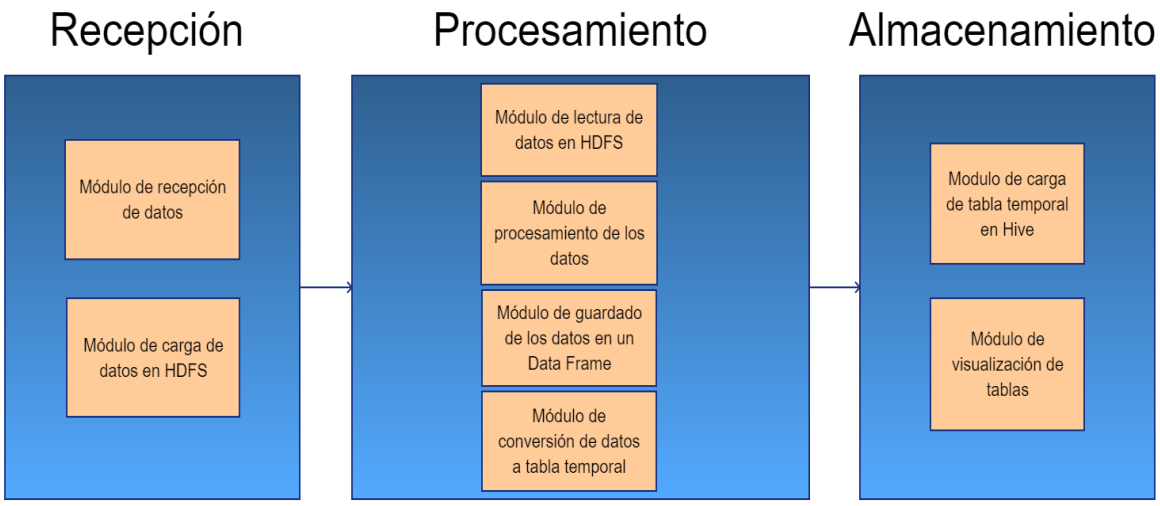


Figura 3-6: Módulos del proyecto

3.3.3 Bases de Datos

Para nuestro proyecto se necesita alguna forma de almacenar nuestros datos de manera estructurada y de más fácil acceso que únicamente con el sistema HDFS. Esto es en esencia una Base de Datos.
Dos de las Bases de Datos más famosas que Apache ofrece son Hive y HBase, las descripciones de éstas se encuentran en el apartado *2.2.2 Bases de Datos* [9].

- **Hive**
  Apache Hive es un data warehouse, que permite realizar consultas más tradicionales. Esto facilitaría que los clientes pudiesen hacer consultas ellos mismos o resultaría más sencillo enseñarles los datos en Hive al ser más similar a lo que se lleva utilizando toda la vida.

- **HBase**
  A diferencia de Hive, HBase es una Base de Datos NoSQL, lo que le da una mejor adaptabilidad a los datos no estructurados y le permite realizar consultas en tiempo real.

3.3.4 Gestor web Apache Ambari

Apache Ambari es una herramienta web-base con la capacidad de crear y administrar clusters. Permite desde elegir qué máquinas se quieren meter en el cluster, hasta qué herramientas se desea instalar en cada máquina, permitiendo así establecer los roles de cada uno de los servidores y poder orientarlos hacia la tarea requerida [10].

3.3.5 Apache Zeppelin

Apache Zeppelin es un web-base notebook que será utilizado a lo largo del trabajo para la realización de los módulos.

Mediante intérpretes de cada lenguaje, Zeppelin dará acceso a Spark y permitirá escribir y ejecutar códigos en Scala y Spark SQL, así como en Python o R si fuese necesario para el proyecto global.

4 Desarrollo

En este apartado se abordaran todos los pasos relacionados con la instalación y creación de las tablas que se usarán para alimentar la futura máquina de estados.

- Instalación del SO
- Instalación de Apache Ambari
- Creación del Cluster
- Preparación de los servicios
- Módulo de carga de los datos en HDFS
- Módulo de lectura y procesamiento de los datos sobre diferentes fuentes
- Módulo de almacenamiento distribuido
- Módulo generador de queries de dominio

4.1 Instalación del Sistema Operativo

Durante el proceso de selección del Sistema Operativo se realizaron búsquedas para cerciorarse de la capacidad de los distintos Sistemas Operativos en la utilización de las herramientas de Apache para Big Data.

Para la selección del Sistema Operativo se han utilizado tres criterios:

1º: Usabilidad y estabilidad
2º: Familiaridad
3º: Instalación de la última versión

La usabilidad y estabilidad de un sistema Big Data, como el de cualquier entorno de producción, es fundamental, por lo que el Sistema Operativo que se seleccionara para ser instalado en los servidores debía ser robusto y fiable.

La primera preselección se llevó a cabo teniendo en cuenta exclusivamente el primer requisito, y se realizó una búsqueda de Sistemas Operativos para servidores con Big Data. Con esta preselección se obtuvieron dos candidatos, Ubuntu Server y CentOS.

CentOS es un Sistema Operativo basado en RedHat y que por lo tanto, dispone de actualizaciones más periódicas y menos constantes que Ubuntu Server. Aunque esto no representa grandes diferencias en el apartado de estabilidad, sí decantó un poco la balanza de la estabilidad a favor de CentOS.

No obstante, ambos sistemas han evolucionado positivamente con el tiempo, ambos son perfectamente capaces de albergar el ecosistema Hadoop y la estabilidad no era el único punto a tener en cuenta, por lo que la elección fue en base a cuál de los dos cumplía mejor con los dos criterios restantes.

En este momento fue cuando el criterio número dos, factor de la familiaridad con el entorno, fue tenido en cuenta. Al no ser la estabilidad razón suficiente para decantarse por ninguno de los dos Sistemas Operativos, la empresa Gain Dynamics prefirió la instalación de la versión para servidores de Ubuntu, puesto que los miembros de la empresa estaban más familiarizados con este SO y la curva de aprendizaje sería mínima.

Por desgracia resultó que esta elección entró en conflicto con el tercer criterio, la instalación de la última versión. Este problema se descubrió durante la realización del apartado *4.3 instalación de Apache Ambari*.

En el momento de la realización de este Trabajo de Fin de Grado no todos los módulos de Apache Hadoop estaban probados y modificados para funcionar en Ubuntu Server 16.04. Actualmente Apache aún está trabajando para llevar todos los módulos de Hadoop a Ubuntu 16.04, y la última versión disponible estable para Ubuntu 14.04 ya está anticuada.

De hecho, en el caso de Apache Ambari, si se pretende instalar la última versión para que sea lo más compatible posible con Ubuntu 16.04, se debe instalar la versión de desarrollador, no la estable.

A día de hoy aún se está en ese punto de cambio del ecosistema Hadoop al nuevo Sistema Operativo de Ubuntu, este hecho unido a la importancia de la instalación de la última versión fue lo que terminó de decantar la balanza a favor de CentOS, que es el Sistema Operativo que se utilizó finalmente.

Dentro de las distribuciones de CentOS se decidió utilizar la versión seis. CentOS 6.8 fue elegido entre las distintas versiones por ser el Sistema Operativo más probado con las versiones exactas de las herramientas necesarias en el proceso de la instalación manual, herramientas como por ejemplo Apache Maven.

De esta forma, se podría realizar una instalación manual de algunos productos de Apache Hadoop si fuese requerido en algún momento en el futuro.

Para la instalación se utilizará la versión Minimal ISO, esta versión permite realizar una configuración de los servidores mucho más limpia. Se evita la mayor parte de librerías y servicios que vienen en los paquetes de instalación más completos.

Este software precargado es útil en la mayoría de los casos, pero dado que este proyecto está tan enfocado en su utilización con herramientas Big Data en exclusivo, resultan añadidos innecesarios para el Sistema Operativo que lo único que hacen es ocupar posibles puertos o iniciar procesos y demonios de programas que no se utilizarán, consumiendo así recursos que pueden llegar a ser cruciales.

Esta versión Minimal también permite manejar la interfaz gráfica y las conexiones de red según las necesidades que se tengan. Por defecto, esta versión no trae ni interfaz gráfica ni conexión automática a internet, lo que permite tanto instalar personalmente la interfaz gráfica y manejar sus recursos como controlar el acceso de los servidores a internet desde el principio y evitar actualizaciones automáticas indeseadas.

Una vez instalada la interfaz gráfica, se puede activar o desactivar en función de las necesidades de control y mantenimiento, y posteriormente desactivarla para evitar que ocupe memoria en el sistema.

La instalación de CentOS no tuvo ningún inconveniente ni se realizó nada fuera de lo común durante el proceso. Cabe mencionar que se formateó toda la memoria de los servidores para la instalación y que no se instaló ningún GRUB. Para un funcionamiento óptimo de Hadoop se recomienda evitar cualquier gestor de particiones o de memoria.

4.2 Preparación del Sistema Operativo

Una vez seleccionado el Sistema Operativo que se iba a utilizar, el siguiente paso fue la preparación de éste para acoger el ecosistema de Apache Hadoop.

Como se explicó en el apartado anterior, la versión que se instaló fue CentOS 6.8 Minimal. Por lo tanto el primer paso fue la habilitación de internet para poder realizar la instalación de Apache Ambari, que fue el gestor que se utilizó para controlar el cluster.

Los servidores de la empresa cuentan con dos entradas Ethernet, em1 y em2. Para configurarlas se deben modificar los archivos */etc/sysconfig/network-scripts/ifcfg-em1* y *ifcfg-em2* respectivamente. Esta primera modificación se debe realizar con el editor "vi", que es el editor de consola que viene instalado por defecto. Una vez conectado internet se podrá instalar el editor que resulte más afín.

Ethernet em1 será el encargado de suministrar conexión a internet y será manejado por el DHCP modificado de la empresa. Ethernet em2 estará configurado con una IP estática y estará conectado a un switch que se encargará de la conexión entre los servidores en red local.

Esta dualidad permite que la información entre servidores de un mismo cluster sea mucho más rápida, alcanzando la velocidad recomendada de 1GB/s. También permite que no se tenga que pasar por el router de la empresa, lo cual lo sometería a una carga extra innecesaria y ralentizaría la red interna.

Para que los servidores se identifiquen entre ellos es necesario modificar el archivo */etc/hosts* y añadir la IP junto con el nombre del resto de servidores que se deben utilizar en el cluster.

Una vez configurado el acceso a la red, el siguiente paso será instalar el servidor de SSH. Esto aporta dos beneficios:

- Permite que los servidores se comuniquen entre ellos sin la necesidad de utilizar una contraseña.
- Permite controlar los servidores en remoto con la herramienta Putty, pudiendo de esta forma realizar las siguientes tareas de configuración desde fuera de la sala de servidores. Putty se conecta por SSH, por lo tanto actúa como una terminal remota.

Se puede instalar openssh client y server desde yum con los repositorios oficiales. Una vez instalado es importante especificar qué niveles tiene activos con el comando "chkconfig sshd on". Estos niveles otorgan diferentes permisos al servicio, desde poder crear un daemon hasta reiniciar o apagar el sistema.

Una vez instalado SSH y configurados los permisos se inicia el servicio con el comando "service sshd start" y se realiza una modificación del archivo de configuración del sistema encargado de los puertos, iptables, añadiendo la restricción de que sólo las máquina que se encuentren en la red local puedan acceder con SSH. De esta forma se evitan posibles accesos externos.

Algo fundamental para el correcto funcionamiento entre grupos de servidores es asegurarse de que todos tienen el reloj del sistema sincronizado de la misma manera y con el mismo reloj de referencia. Esto se consigue instalando el servicio NTP.

Al igual que en el caso del SSH, NTP puede descargarse de yum y necesita de los permisos que otorga el comando "chkconfig ntpd on". Es aconsejable que los relojes de referencia que vienen por defecto con la instalación sean cambiados por los relojes más cercanos geográficamente. Esto se modifica con el comando "vi /etc/ntp.conf". Tras el cambio se actualizan todos los servidores a la vez y se realiza un restart del servicio.

Durante el proceso de instalación de Apache Ambari o cualquier otra herramienta de Hadoop es aconsejable que se desactive cualquier mecanismo de control del acceso. Esto incluye desactivar iptables y SELinux.

Para desactivar iptables es necesario desactivarlo del chkconfig y parar el servicio con el comando "/etc/init.d/iptables stop". En cambio para detener SELinux hay que modificar su archivo de configuración con el comando "/etc/selinux/config" y cambiar el valor SELINUX a disabled.

Si bien iptables puede acarrear problemas principalmente de conexiones entre nodos, SELinux puede llegar a producir un mal funcionamiento de Apache Hadoop y provocar errores internos, aunque es posible crear una excepción para los servicios de Hadoop nada asegura que todo funcionará correctamente. Una vez terminado el setup, iptables, a diferencia del SELinux, sí que se puede volver a activar.

Otro servicio que hay que desactivar es Transparent Huge Pages o THP por sus siglas en inglés. Como se ha comentado anteriormente en esta memoria, una de las bases de Apache Hadoop es HDFS, el sistema de archivos distribuidos. El objetivo de Transparent Huge Pages es modificar en función de las necesidades el tamaño de los bloques de memoria, también llamados páginas, para así conseguir reducir el número de páginas por programa.

Este comportamiento es beneficioso tanto para el almacenamiento de los datos como para la ejecución de un programa. El problema es que Hadoop ya tiene su propio sistema de archivos y estas modificaciones dificultan el almacenamiento distribuido.

La manera de deshabilitar esta función es durante el encendido. Se modifica el archivo */etc/rc.local* y se añaden estas dos sentencias:

- if test -f /sys/kernel/mm/redhat_transparent_hugepage/enabled; then
  echo never > /sys/kernel/mm/redhat_transparent_hugepage/enabled
  fi

- if test -f /sys/kernel/mm/redhat_transparent_hugepage/defrag; then
  echo never > /sys/kernel/mm/redhat_transparent_hugepage/defrag
  fi

Estas sentencias deshabilitan THP siempre que esté iniciado al arrancar y de esta forma se evita que se reactive por accidente durante alguna actualización o instalación.

Con el objetivo de poder realizar la descarga de Ambari a través de un repositorio se utilizará el commando wget. Este comando permite descargar archivos con la terminal y puede instalarse sin problemas a través de yum ya que se encuentra en los repositorios oficiales.

El último paso de configuración del Sistema Operativo es relativo a la instalación de una Base de Datos para Hive. Hive necesita una Base de Datos preinstalada que utilizar, aunque no es el único producto de Apache Hadoop que requiere una BD, si es el que influye más en la elección de qué Base de Datos usar.

De todas las Bases de Datos compatibles con Hive las más utilizadas son MySQL y PostgreSQL. Para la elección entre estas dos Bases de Datos nos vamos a fijar en sus principales atributos.

PostgreSQL es una Base de Datos estable, robusta y con una gran escalabilidad, todos ellos factores muy a tener en cuenta para Big Data. No obstante y pese a no ser una Base de Datos tan robusta como PostgreSQL, MySQL posee varias propiedades que decantan la balanza a su favor.

MySQL es una Base de Datos ligera a diferencia de PostgreSQL que consume gran cantidad de recursos, esto es especialmente importante cuando el sistema está ejecutando varias tareas a la vez. Además MySQL cuenta con una mayor velocidad de procesamiento a la hora de realizar operaciones.

La Base de Datos que se ha instalado finalmente ha sido MySQL, para ello fue necesario acceder a los repositorios de CentOS, descargarlo con yum mediante el comando “yum install mysql-server” e iniciar el servicio con “/sbin/service mysqld start”.

Se realizó la configuración de seguridad de la Base de Datos a través de la ejecución de “/usr/bin/mysql_secure_installation”.

Debido a diversos requisitos de la empresa también se ha instalado una interfaz gráfica. Los siguientes tres comandos son los utilizados para este propósito:

- yum -y groupinstall "X Window System"
- yum -y groupinstall "Desktop"
- yum -y groupinstall "General Purpose Desktop"

Para activar la interfaz es necesario ejecutar startx en la terminal de la máquina. Puesto que Putty utiliza SSH no se puede conectar a un escritorio gráfico. Por lo tanto también se ha configurado la conexión remota y se acceden a los servidores mediante VCN viewer.

4.3 Instalación de Apache Ambari

Tras la preparación llevada a cabo en el apartado anterior ya se puede instalar Apache Ambari.

El primer paso es añadir el repositorio con la versión exacta que se desea, en nuestro caso la última estable disponible. Para ello se utiliza el comando:
"wget -nv http://public-repo1.hortonworks.com/ambari/centos6/2.x/updates/2.4.2.0/ambari.repo -O /etc/yum.repos.d/ambari.repo"

Una vez añadido el repositorio, se actualiza y se comprueba que los paquetes se han añadido correctamente.

Como se explicó en el apartado *3.3.4 Gestor web Apache Ambari*, se utiliza Ambari para controlar y crear el cluster. Apache Ambari consta de dos partes:

- Apache Ambari server: Este servicio es el encargado de monitorizar el estado de los servidores y de enviarles instrucciones a través de los agent. Por este motivo, sólo se instala un server por cluster, generalmente en uno de los servidores master.

- Apache Ambari agent: Los agent son los encargados de enviar y recibir la información del Ambari server. Recaban la información y ordenan a los diferentes nodos de Hadoop qué tareas hacer. Por lo tanto y en contraposición al server, Ambari agent se instala por servidor.

Ahora ya se está preparado para instalar Apache Ambari server, se ejecuta el comando con "yum install ambari-server" y se instalan los paquetes.

```
rver-8.4.20-7.el6.x86_64
--> Processing Dependency: libpq.so.5()(64bit) for package: postgresql-server-8.4.20-7.
el6.x86_64
--> Running transaction check
---> Package postgresql.x86_64 0:8.4.20-7.el6 will be installed
---> Package postgresql-libs.x86_64 0:8.4.20-7.el6 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package              Arch       Version             Repository                  Size
================================================================================
Installing:
 ambari-server        x86_64     2.4.2.0-136         Updates-ambari-2.4.2.0     645 M
Installing for dependencies:
 postgresql           x86_64     8.4.20-7.el6        base                       2.6 M
 postgresql-libs      x86_64     8.4.20-7.el6        base                       202 k
 postgresql-server    x86_64     8.4.20-7.el6        base                       3.4 M

Transaction Summary
================================================================================
Install       4 Package(s)

Total download size: 652 M
Installed size: 729 M
Is this ok [y/N]: y
```

Figura 4-1: Install Ambari server

En el servidor donde está funcionando Ambari server se instala el conector de la Base de Datos, “yum install mysql-connector-java” y se da permisos con “chmod 644 /usr/share/java/mysql-connector-java.jar”.

Después, se conecta Ambari server con mysql-connector para establecer la Base de Datos como MySQL con la instrucción “ambari-server setup --jdbc-db=mysql --jdbc-driver=/usr/share/java/mysql-connector-java.jar”

Una vez hecho esto, se accede a MySQL por terminal y se crean los usuarios que se necesiten y sus correspondientes Bases de Datos. Para este proyecto, se han creado tres usuarios y tres Bases de Datos, una para Ambari, otra para Hive y otra para Oozie, mediante los siguientes comandos:

- CREATE USER 'ambari'@'%' IDENTIFIED BY '****';
- GRANT ALL PRIVILEGES ON *.* TO 'ambari'@'%';
- FLUSH PRIVILEGES;
- CREATE DATABASE ambari;

Para la creación de la Base de Datos de Hive y Oozie, se repiten las cuatro instrucciones anteriores cambiando ambari por hive u oozie.

Tras la preparación del conector es necesario realizar la configuración del servicio, se ejecutan “ambari-server setup” y se siguen los pasos de la configuración respondiendo a las preguntas del configurador.

```
Configuring database...
============================================================================
Choose one of the following options:
[1] - PostgreSQL (Embedded)
[2] - Oracle
[3] - MySQL / MariaDB
[4] - PostgreSQL
[5] - Microsoft SQL Server (Tech Preview)
[6] - SQL Anywhere
[7] - BDB
============================================================================
Enter choice (1): 3
Hostname (localhost):
Port (3306):
Database name (ambari):
Username (ambari):
Enter Database Password (bigdata):
Configuring ambari database...
Copying JDBC drivers to server resources...
Configuring remote database connection properties...
WARNING: Before starting Ambari Server, you must run the following DDL against t
he database to create the schema: /var/lib/ambari-server/resources/Ambari-DDL-My
SQL-CREATE.sql
Proceed with configuring remote database connection properties [y/n] (y)? y
Extracting system views...
.....ambari-admin-2.4.2.0.136.jar

Adjusting ambari-server permissions and ownership...
Ambari Server 'setup' completed successfully.
```

Figura 4-2: Config Ambari server

Para este proyecto se realizó la instalación de las últimas versiones de los productos necesarios para la configuración de Ambari server como JDK y se estableció una contraseña para el servicio.

Una vez terminada la configuración se inicia el server con “ambari-server start”.

Desde este instante Apache Ambari server ya está activo y es accesible a través del navegador con ruta “IP_Host_Ambari_server:8080”. Esta ruta direccionará a la ventana de login donde introducir usuario y contraseña. En cuanto se pueda es recomendable cambiar estos datos de fábrica a unos personalizados mediante las herramientas que proporciona Ambari.

Con Ambari server ya en línea, hay que preparar Ambari agent, uno por máquina.
Primero se instala el mismo repositorio que se ha utilizado para instalar Apache Ambari server en el resto de máquinas. En el caso de este Trabajo de Fin de Grado la empresa Gain Dynamics disponía de dos servidores, por lo que sólo hacía falta instalar el repositorio en el servidor que faltaba.

Tras comprobar que los paquetes están bien instalados, se procede a instalar los agentes. Este proceso es similar al de instalación del servidor. Se utilizó yum para ejecutar el comando “yum install ambari-agent”.

Al terminar la instalación de los agentes es necesario indicarles la localización del Ambari server con el que se van a estar comunicando. Para esto se modifica el archivo “ambari-agent.ini” que se encuentra en la ruta “/etc/ambari-agent/conf/”.

En el archivo “ambari-agent.ini” se cambia “hostname=localhost” por el nombre del servidor en el que se ejecuta Ambari server, el mismo nombre que se utiliza para conectar los servidores en la modificación del archivo */etc/hosts* en el apartado anterior.

Tras estos pasos se inicia el agente con la instrucción “ambari-agent start”.

4.4 Creación del Cluster

La creación del cluster es el momento en el que el ecosistema Hadoop toma forma. Durante este proceso se instalaran los diferentes módulos de Hadoop y se establecerán los enlaces necesarios.

El primer paso es acceder a Ambari con el navegador y loguearse.

Una vez logueados, se pulsa el botón de crear cluster y aparecerá el asistente de instalación del cluster.

Como se puede ver en la figura 4-3, este asistente consta de once pasos, a lo largo de este apartado se detallará cada uno de ellos.

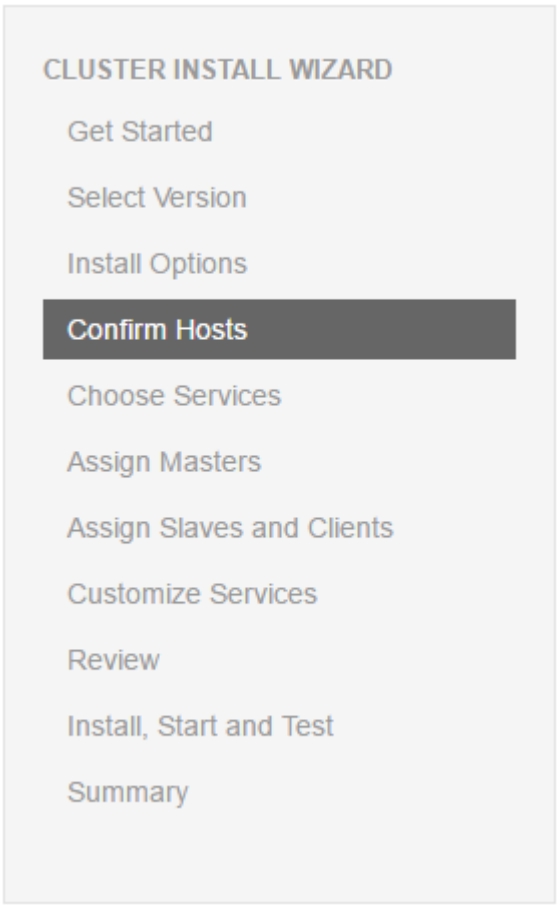


Figura 4-3: Install Wizard

Get Started no contiene nada de información ni configuración, por lo que el primer paso real es Select Version

En Select Version, figura 4-4, se debe seleccionar qué versión de Hadoop se desea instalar. Puesto que uno de los requisitos de este proyecto y uno de los motivos por el cual se decidió CentOS como Sistema Operativo era instalar la última versión disponible, se descarga la última versión HDP 2.5 de los repositorios públicos.

Se elimina de la lista de repositorios todas las versiones de Sistemas Operativos que no sean redhat6. Tanto para Red Hat como para CentOS el repositorio es el mismo, dado que uno se basa en el otro, lo cual los hace casi idénticos. De hecho en la URL de redhat6 aparece centos6.

Figura 4-4: Select Version

Con el repositorio y la versión elegidos se accede al siguiente paso, figura 4-5, las opciones de instalación. En este paso se introduce el Fully Qualified Domain Name de cada servidor y se selecciona realizar un registro manual de los servidores.

Se recomienda el registro manual por razones de seguridad.

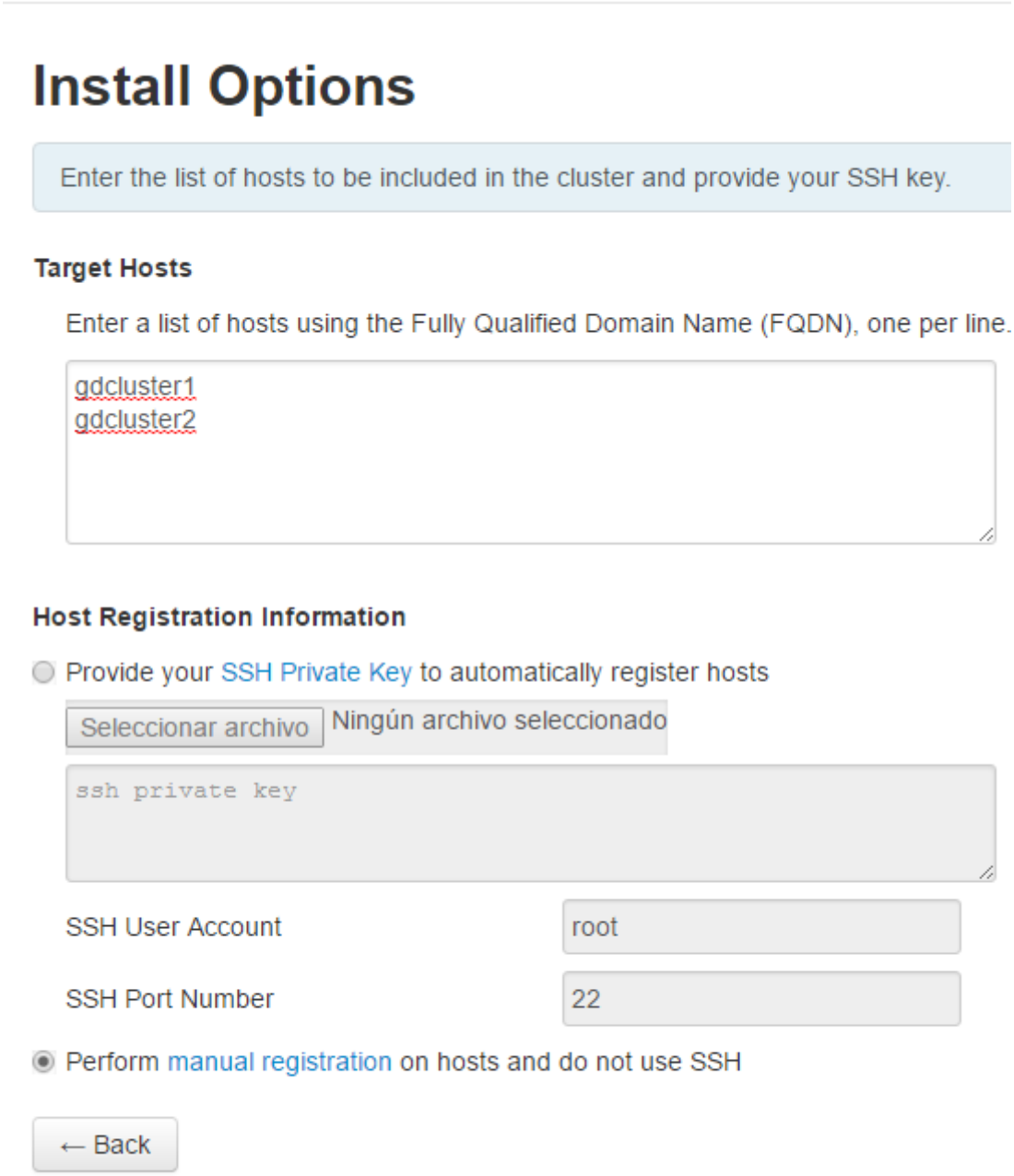


Figura 4-5: Install Options

Al pulsar en siguiente saldrá una advertencia indicando que los Ambari agent deben estar instalados en cada servidor. Se pulsa ok y se continúa.

El siguiente paso se encarga de la confirmación del buen estado de los servidores especificados en el paso anterior. Una vez la comprobación finalice nos mostrará si hay algún problema de conexión o algún otro problema con Firewall, JDK, etc.

Con los servidores correctamente configurados llega el momento de la selección de las herramientas de Apache Hadoop que se desean instalar en los servidores.

Esta acción se realiza en el paso Choose Services, en el cual aparecen una serie de productos de Hadoop para elegir con checkbox junto a una breve descripción de lo que hacen.

De todos los productos de la lista a priori solo se iban a instalar los básicos, pero durante el desarrollo de este Trabajo de Fin de Grado se decidió instalar más servicios pensando en el futuro uso de estos servidores y en el proyecto global de Gain Dynamics.

Una vez seleccionados qué elementos se desean, se accede al siguiente paso, figura 4-6, la asignación de los masters. En este paso se asigna en qué servidor estará cada master de cada componente.

Puesto que disponemos de dos servidores, lo correcto es balancear la carga entre los dos. De esta forma se evita tener un cuello de botella producido por tener un servidor que sólo controla y otro que sólo ejecuta.

De todos estos componentes ZooKeeper es el único que debe instalarse en todos los servidores, porque Ambari lo utiliza para enviar y recibir información.

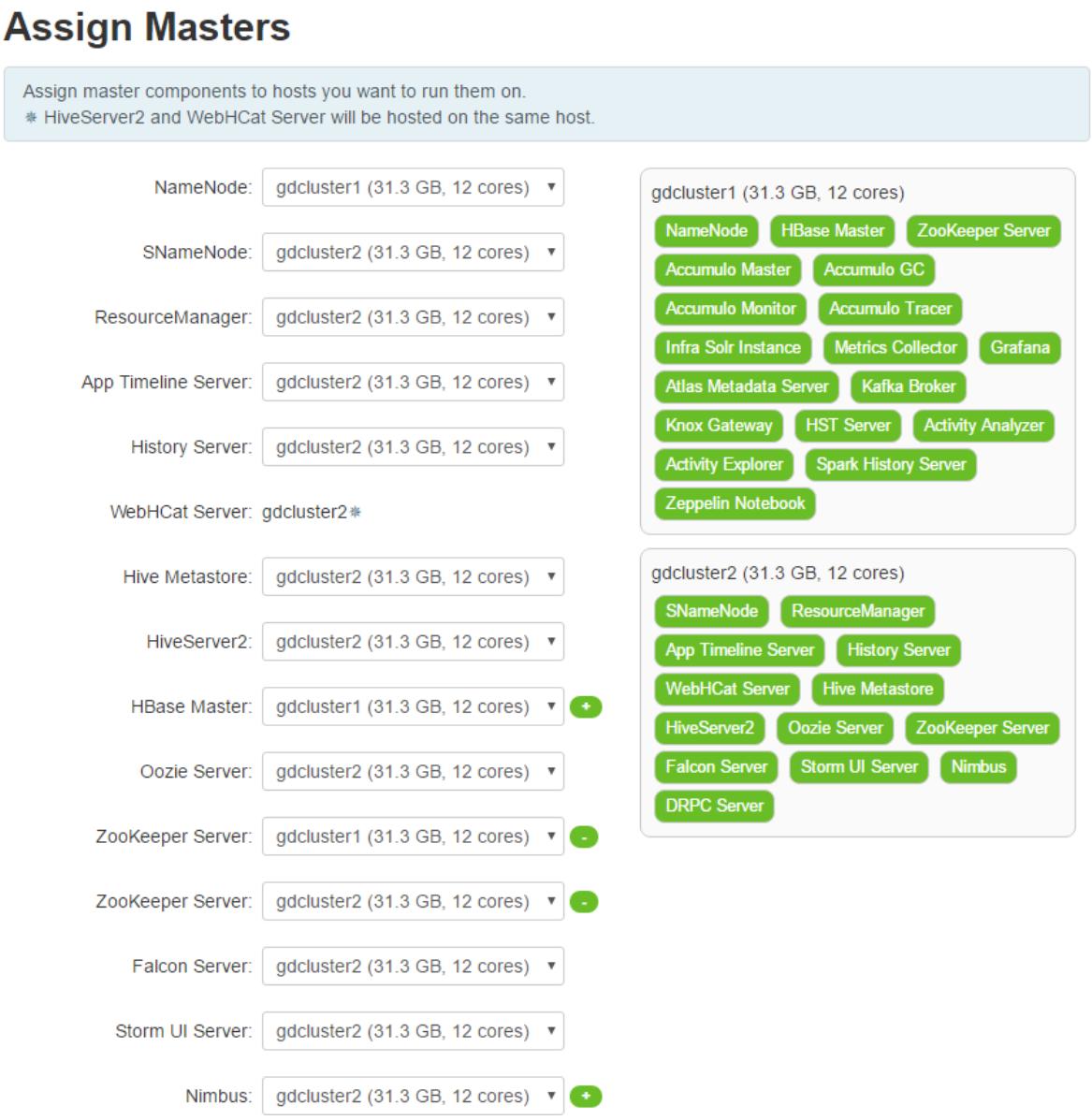


Figura 4-6: Assign Masters

Tras asignar los masters hay que asignar los slaves, esto se realiza en el siguiente paso Assign Slaves and clients. Mediante un checkbox se seleccionan qué componentes de slave se desean instalar en cada servidor.

Al igual que en el paso anterior, se distribuirá la carga entre los dos servidores.

Una vez elegido, se presiona siguiente y se entra en el paso Customize Services. Como puede verse en la figura 4-7 para el caso de Oozie, en este paso se establecerán los límites de memoria, de disco duro, se establecerán contraseñas para servicios o qué Base de Datos se utilizará para un servicio concreto.

Para las contraseñas y Bases de Datos se introducirán los datos pertinentes en los campos requeridos. En cuanto a la limitación o ampliación de recursos, Ambari ya hace una preselección bastante acertada de los valores a utilizar. Tan sólo se aumentaron un poco los valores de HDFS.

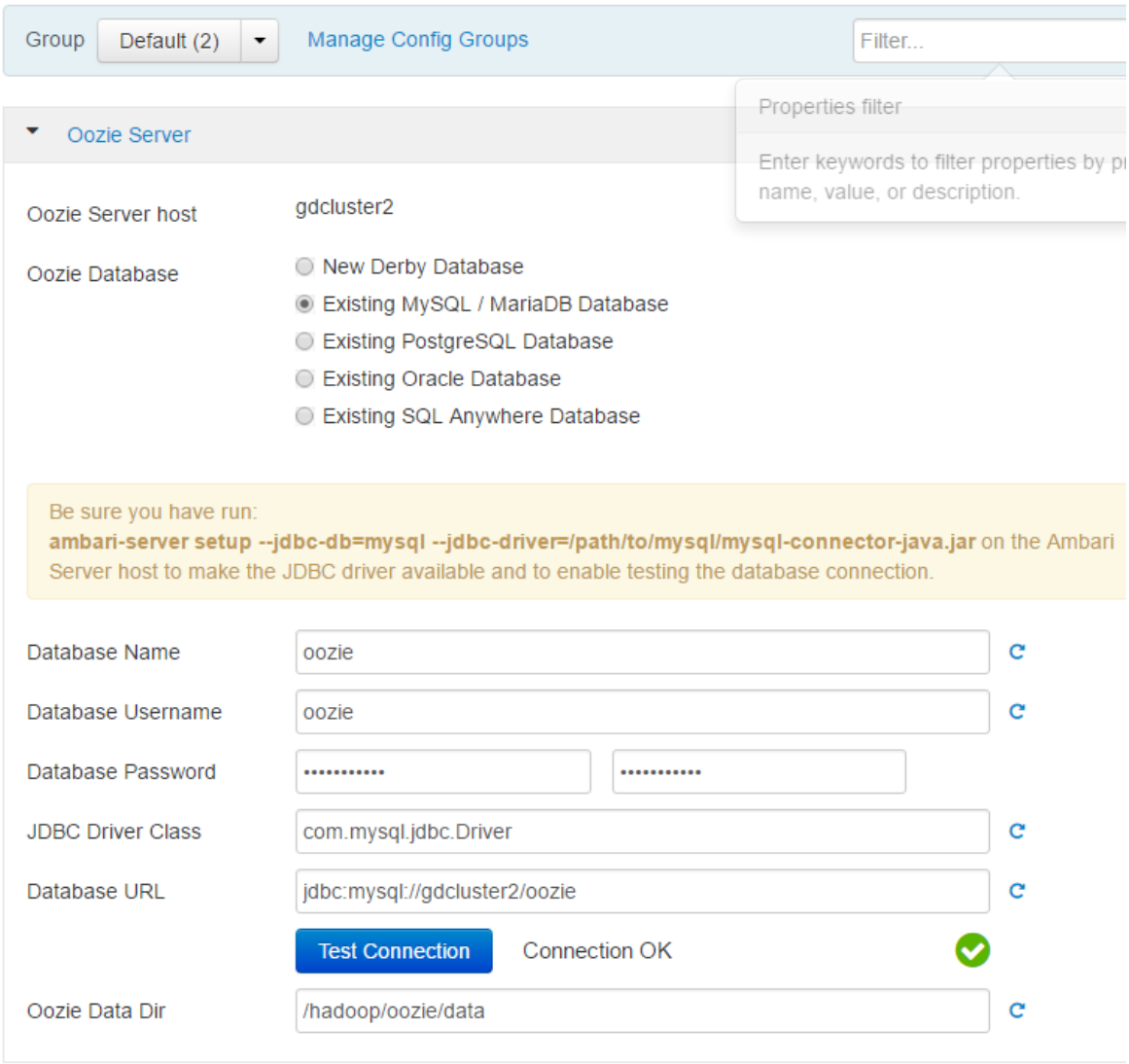


Figura 4-7: Customize Services

Con los servicios ya configurados la instalación está ya casi completa. Ya sólo resta ver el siguiente paso, Review, que como su nombre indica presenta todos los datos de la configuración que se han realizado y permite comprobar que todo está bien.

Se presiona el botón de deploy y se accede al penúltimo paso, Install,Start and Test.
Este paso se encarga de la instalación de todos los elementos que se han seleccionado y configurado. La figura 4-8 es una muestra del proceso.

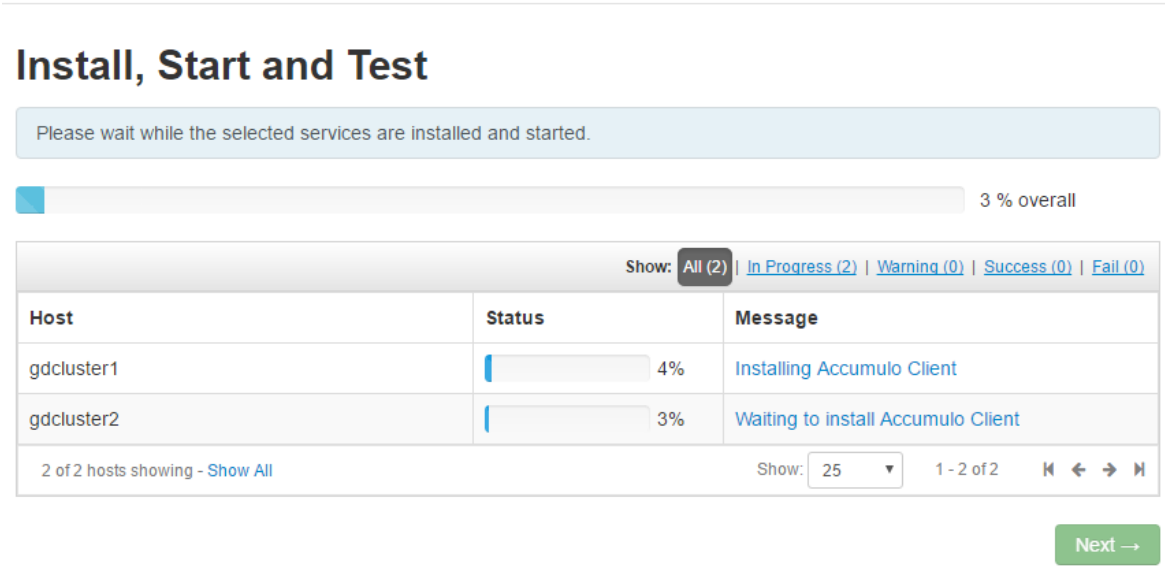


Figura 4-8: Install

Una vez termina la instalación se pasa Summary, el paso final en el cual se muestra cómo ha ido la instalación.

Como puede apreciarse en la figura 4-9, los servicios han fallado a la hora de autoiniciarse, esto es debido a que aún quedan preparaciones que hacer para los servicios recién creados.

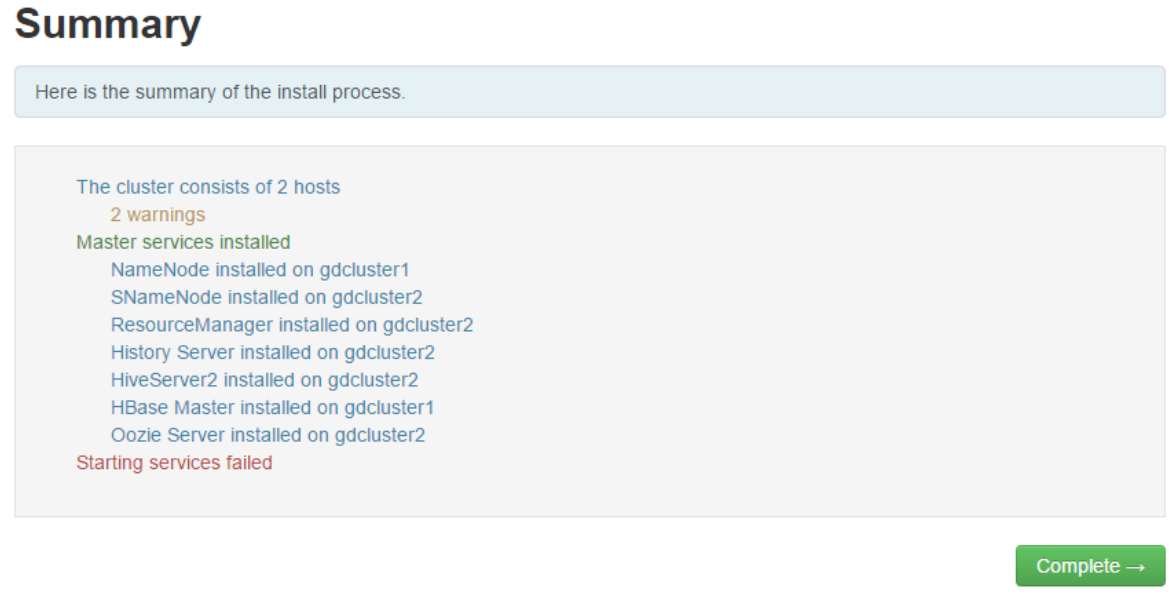


Figura 4-9: Summary

4.5 Preparación de los servicios

Antes de iniciar los servicios hay un par de cosas que aún se deben hacer para poder acceder a todo lo que ofrece Apache Hadoop.

Primero se deben dar permisos a Ambari para acceder y modificar HDFS como si de un editor se tratase. Para ello se ejecutan estos cuatro comandos:

- sudo su - hdfs
- hdfs dfs -mkdir /user/admin
- hdfs dfs -chown root:hdfs /user/admin
- exit

De esta forma se crea una carpeta con el usuario de Apache Ambari, en nuestro caso admin, y le da permisos para modificar.

Otro paso importante es añadir el servidor que está ejecutando Ambari en el archivo hosts del sistema.

Si el SO es Linux, el procedimiento es el mismo que se explica en el apartado *4.2 Preparación del Sistema Operativo*, se modifica el archivo */etc/hosts* y se añade la IP junto con el nombre de servidor.

En caso de que el SO sea Windows, para evitar problemas con permisos, el procedimiento es el siguiente:

- Acceder a la ruta *C:\Windows\System32\drivers\etc*
- Copiar el archivo *hosts* al escritorio y abrilo con notepad++ u otro editor similar
- Añadir la IP con el nombre del servidor, en nuestro caso 192.168.1.203 gdcluster1
- Copiar el archivo *hosts* de vuelta a *C:\Windows\System32\drivers\etc*

Con este cambio el equipo será capaz de acceder a los datos del notebook Apache Zeppelin.

Hechos todos estos pasos ya se pueden iniciar los servicios. Se accede a Ambari a través del navegador, nos logueamos y en el Dashboard, debajo de todos los servicios en el desplegable de Actions, se pulsa Start all. Esto iniciará todos los servicios a la vez.

Una vez los servicios estén en línea el Dashboard empezará a recibir datos de uso.

4.6 Módulo de carga de datos en HDFS

Una de las ventajas de Ambari para el módulo de carga de datos es que permite realizar tareas como la gestión de archivos de manera muy fácil y rápida. Para cargar los datos en HDFS basta con abrir la ventana de Files View y pulsar en el botón de Upload, figura 4-10.

También puede realizarse ejecutando el script del módulo de carga de datos. Este script se creó con el intérprete Shell de Zeppelin, y realiza labores de carga y control de los datos almacenados en HDFS por terminal.

De esta forma se puede utilizar por ejemplo el comando "hdfs dfs -put /root/Documents/cuantificadores.csv /tmp/cuantificadores.csv" para cargar datos en HDFS.

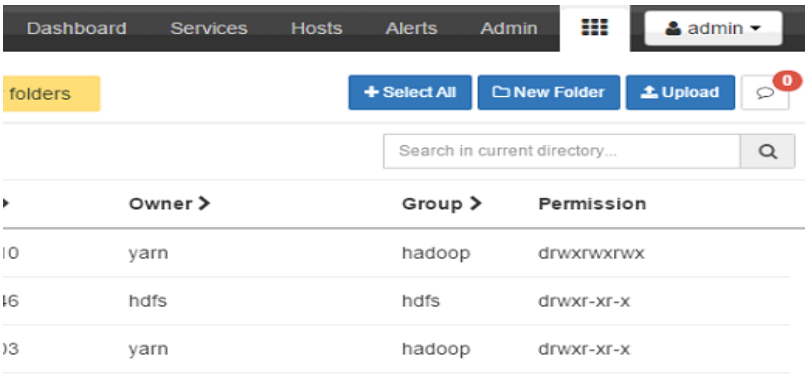


Figura 4-10: Upload HDFS

4.7 Módulo de lectura y procesamiento de los datos sobre diferentes fuentes

Una vez que ya se tienen los datos cargados en HDFS es el momento de utilizar la potencia que nos otorga Apache Spark para la lectura y procesamiento de datos [11]. Las imágenes de este apartado han sido tomadas de diferentes scripts del módulo de lectura y procesamiento.

El primer objetivo es leer los datos que están almacenados en HDFS, para ello se utiliza textFile y se cargan los datos en variables estáticas. Como aparece en la figura 4-11.

```
val tmpNQZ1 = sc.textFile("/tmp/GLB_GD_CreacionIndicadores_IndicadoresNQ_RasgosZ1.csv")
val tmpNQZ2 = sc.textFile("/tmp/GLB_GD_CreacionIndicadores_IndicadoresNQ_RasgosZ2.csv")

case class Comparadores(
p1: Double,
p2: Double,
p3: Double,
p4: Double,
p5: Double,
p6: Double,
p7: Double,
p8: Double,
p9: Double,
p10: Double,
p11: Double
)
```

Figura 4-11: Read HDFS

Con los datos en las variables estáticas se crea una clase que va a almacenar los datos que se extraigan de los archivos.

```
val NQZ1 = tmpNQZ1.filter(!_.startsWith("#")).map{s=>
val r = s.replaceAll(",",".").split(";")

p1 = r(0).trim().toDouble
p2 = r(5).trim().toDouble+r(6).trim().toDouble
p3 = r(2).trim().toDouble
p4 = r(3).trim().toDouble
```

Figura 4-12: Process data

Una vez extraídos los datos de HDFS se procesan como si de un parser se tratase. Tal y como aparece en la figura 4-12. El módulo contiene varios scripts para procesar varios tipos de archivos y varios tipos de datos.

4.8 Módulo de almacenamiento distribuido en Hive

Una vez procesados los datos, éstos se pueden introducir en un DataFrame. Una vez dentro del DataFrame, Spark permite de una manera muy sencilla transformar ese DataFrame en una tabla temporal. Como se aprecia en un script de este módulo, figura 4-13.

```
NQZ1.registerTempTable("NQZTbl")
```

Figura 4-13: Create TempTable

Esta tabla temporal puede utilizarse como cualquier otra tabla estándar, permitiendo así realizar un insert en la tabla y crear una tabla dentro de Hive de manera fija como muestra la figura 4-14, que representa unos de los scripts de SQL creado en Zeppelin para este módulo.

```
%sql
CREATE TABLE GLB_GD_CreacionIndicadores_IndicadoresNQ_transPR
AS SELECT a.*, b.*
FROM NQZTbl a
LEFT OUTER JOIN NQTraspTbl b
ON (a.p1 = b.t1)
```

Figura 4-14: Create DefinitiveTable

4.9 Módulo generador de queries de dominio

Utilizando los scripts del módulo de lectura y procesamiento, se puede crear un script que permita leer fechas e indicadores de cualquier tipo y transformarlos en dos tablas temporales, una transaccional y otra contextual.

Tablas temporales que se podrán pasar a estáticas y ser utilizadas para crear la máquina de estados y proporcionarla datos de entrada.

```
c5 = c5*100/total
c6 = c6*100/total
c7 = c7*100/total
c8 = c8*100/total
c9 = c9*100/total

contador = contador+1

d1 = DateTime.parse(c1, DateTimeFormat.forPattern("dd/MM/yyyy' 'HH:mm"))

mili=d1.getMillis() - d2.getMillis()
mili=mili/3600000
```

Figura 4-15: Procesamiento del timestamp

En la figura 4-15 se puede ver cómo se realiza el tratamiento de los timestamps. Este método permite procesar la variable de tiempo independientemente de cuál sea el formato en el que se reciba, flexibilidad sumamente importante por la variedad de estilos de fechas.

5 Integración, pruebas y resultados

En este capítulo se detallarán las pruebas realizadas para comprobar el funcionamiento de todos los componentes que integran este proyecto.

Las pruebas no se han realizado sobre las tablas definitivas que contendrán la información del cliente, ya que no se disponen de ellas, sino que son pruebas del buen funcionamiento de los elementos instalados con distintas fuentes simuladas para abarcar el mayor número de posibilidades.

5.1 Pruebas CentOS

Las pruebas realizadas al Sistema Operativo han estado enfocadas en la comprobación de las conexiones entre los servidores y la conexión del servidor host de Ambari con otros equipos de la oficina.

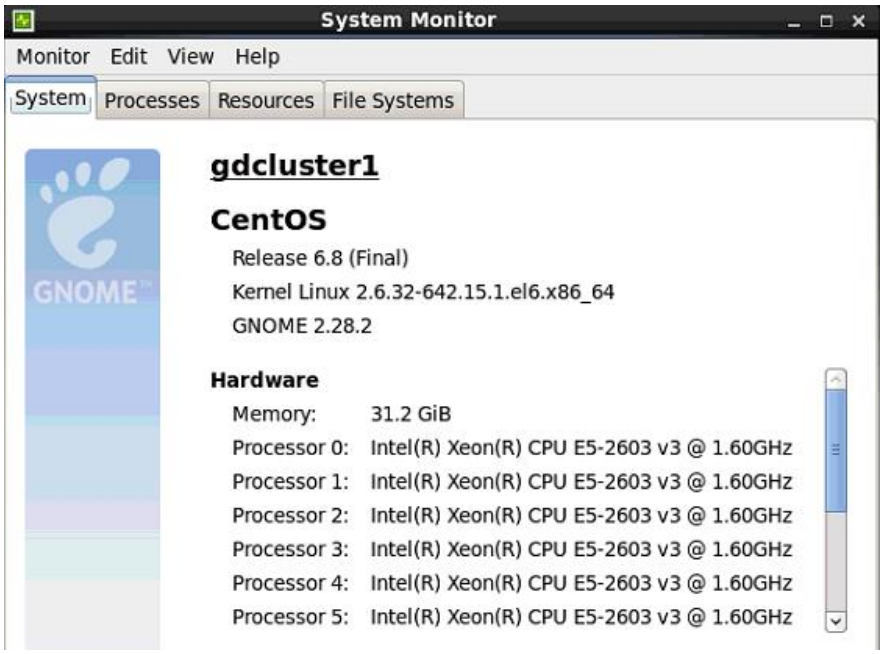


Figura 5-1: CentOS System

Como se puede ver en la figura 5-1, CentOS 6.8 es la versión instalada.

5.2 Pruebas Apache Ambari y servicios

Las pruebas a Apache Ambari constan de tres partes:

- **Asegurar el correcto despliegue de Ambari y los servicios**
  La comprobación del correcto despliegue se basó en el análisis de los diferentes logs de los servicios y en el apoyo visual que aporta Ambari.

- **Comprobar el envío y recepción de datos**
  Para asegurar el envío y recepción de información a los servicios y entre ellos se realizaron diversas acciones como modificar el estado de los servicios o cambiar sus parámetros de configuración y confirmar la realización del cambio.

  También se realizaron comprobaciones relativas a la actividad de los puertos que utilizan los servicios.

- **Pruebas unitarias**
  Las pruebas unitarias se realizaron sobre todos los servicios necesarios para el desarrollo del generador de queries de dominio. Estas pruebas abarcaron desde la realización de tests de subida de archivos a HDFS, hasta la comprobación del correcto funcionamiento del notebook Zeppelin y los intérpretes que utiliza para poder ejecutar los distintos lenguajes.

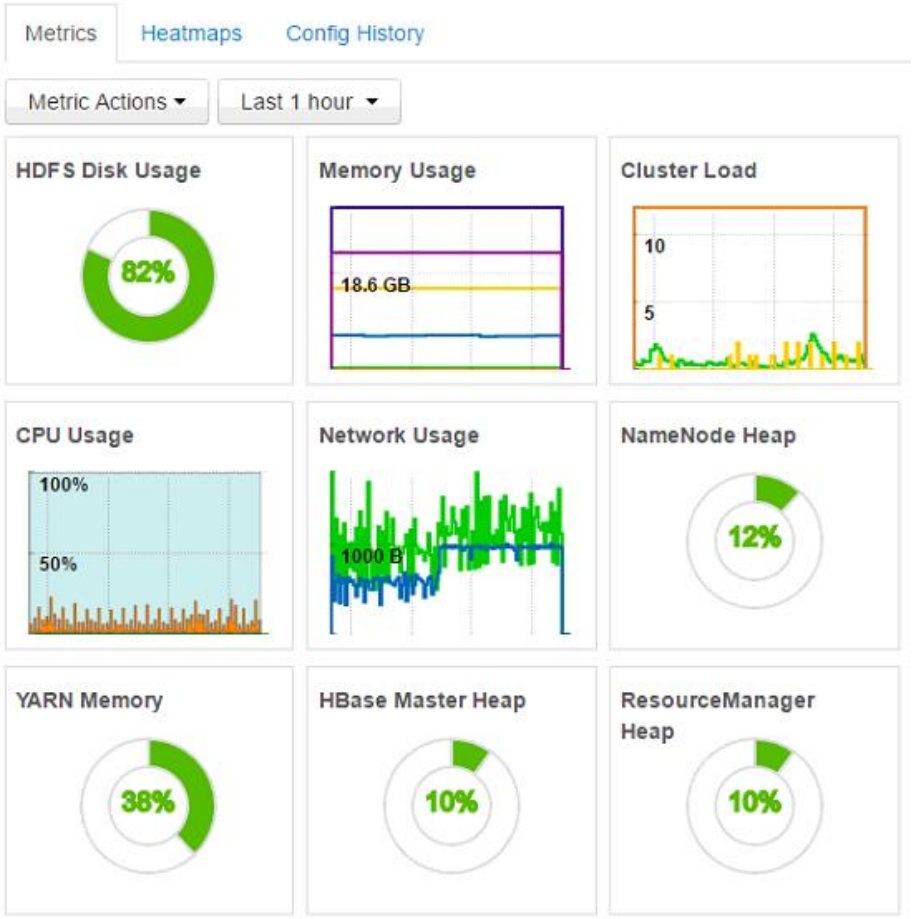


Figura 5-2: Ambari Dashboard

En la figura 5-2 se muestra el Dashboard de Ambari, que contiene los datos en tiempo real de los servicios activos, lo cual certifica el correcto despliegue de servicios y su comunicación.

Un producto que no se utilizó en este proyecto, pero que si se empleará en el proyecto global de la empresa es Hbase.

Hbase no dispone de un buen visor oficial para Ambari, por lo tanto se han realizado las pruebas de creación, manipulación y borrado a través de su visor de terminal, como aparece en la figura 5-3.

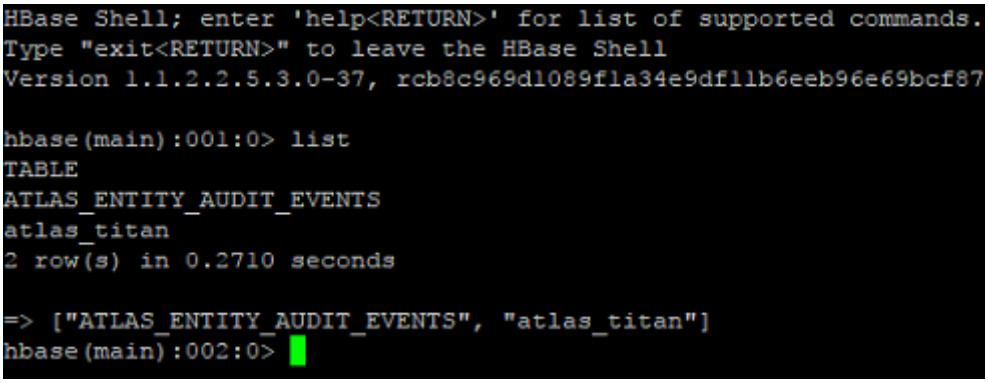


Figura 5-3: Hbase terminal

5.3 Integración de HDFS y Hive con Spark

La conexión entre HDFS y Hive con Spark se realiza utilizando los intérpretes de Apache Zeppelin. La manipulación de datos en HDFS con el intérprete por defecto, Scala, y la creación y tratamiento de tablas en Hive con el intérprete %sql.

En la figura 5-4 se muestra un fragmento de un script del módulo de lectura y procesamiento. En este fragmento se realiza la lectura con Spark de un fichero en HDFS que se utilizará para la creación de las tablas transaccional y contextual.

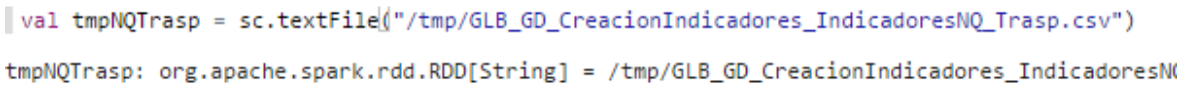


Figura 5-4: HDFS spark

La figura 5-5 contiene la lista de tablas que se encuentran en Hive. Esta lista también indica si las tablas son temporales o no.

En esta figura se puede observar cómo funciona el intérprete de Hive en Zeppelin y cómo las tablas temporales que ha creado Spark mediante el método mostrado en la figura 4-14, pueden manipularse como cualquier otra tabla en Hive.

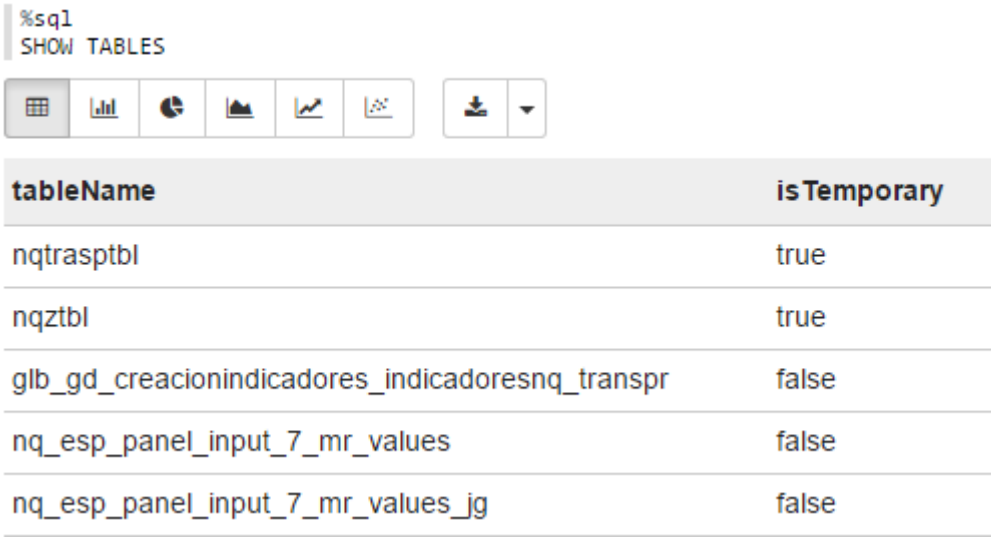


Figura 5-5: Hive SQL

5.4 Resultados del generador de queries de dominio

Para el desarrollo del generador se ha creado un conjunto de scripts unidos en un notebook de Zeppelin. Estos scripts contienen varias formas de procesar diferentes entradas para obtener la salida esperada, de forma que sea lo más automático posible.

Entre las posibilidades que se plantean está unir tablas partidas, procesar tablas con otras, procesar cadenas con caracteres no válidos, realizar manipulaciones a formatos de fechas para almacenar y medir la distancia temporal independientemente de cómo las envíe el cliente, etc.

En la figura 5-6, se aprecia cómo se manipulan dos fechas con distintos formatos y se calcula la distancia temporal entre ambas fechas para poder obtener un valor de distancia entre la toma de decisiones.

Este dato permitirá crear un ciclo de vida del cliente mucho más preciso y de este modo la máquina de estados podrá predecir mejor a que estado saltará en el futuro.

```
import org.joda.time.format._
import org.joda.time._

val d1 = DateTime.parse("05/03/2015 02:01:01", DateTimeFormat.forPattern("dd/MM/yyyy' 'HH:mm:ss"))
val d2 = DateTime.parse("2015/03/04 01:01:01", DateTimeFormat.forPattern("yyyy/MM/dd' 'HH:mm:ss"))
val mili=d1.getMillis() - d2.getMillis()
val segundos=mili/1000
val minutos=segundos/60
val horas=minutos/60
```

Figura 5-6: Date

La figura 5-7 representa la tabla transaccional creada a partir del procesamiento de los datos con Spark. Esta tabla contiene los campos primarios de los que dispondrá la máquina de estados para crearse y alimentarse.

Estos campos son:

- **Fecha**: obtenida de un timestamp u otro formato de tiempo para indicar el momento exacto de actualización de los datos debido a las nuevas acciones del cliente.
- **Tiempo**: Una medida de distancia temporal, en horas, entre cada actualización de los datos.
- **Lista de patrones de navegación**: Una sucesión de columnas con los indicadores de conducta en la red.
- **Lista de operaciones**: Una sucesión de columnas que indican las operaciones que el cliente ha realizado.

Fecha	Tiempo	Orden	Business	Adult	Science
16/06/2016 0:00	0	1	54.24084	6.38743	20.73298
17/06/2016 0:00	24	2	50	26.6791	0
18/06/2016 0:00	48	3	50	0.9901	0
21/06/2016 0:00	120	4	52.28149	5.10925	13.75321
22/06/2016 0:00	144	5	53.4918	5.82667	5.7846
23/06/2016 0:00	168	6	51.21951	13.41463	0
26/06/2016 0:00	240	7	50.49447	3.37405	0
29/06/2016 0:00	312	8	50	15.83333	0

Figura 5-7: Tabla transaccional

Este módulo también genera la tabla contextual, cuyo contenido son los datos sociodemográficos de cada cliente. El propósito de esta tabla es el de posicionar al cliente dentro de factores desconocidos para la empresa como pueden ser su edad, su nivel socioeconómico o el número de hijos. Consiguiendo así una caracterización más personal del cliente.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Tras la realización del proyecto, se puede concluir que se han alcanzado los objetivos propuestos en el inicio de este Trabajo de Fin de Grado.

El objetivo principal de este proyecto era el desarrollo parcial de un sistema que generalice la inferencia de máquinas de estado (o variantes suyas) para que, a partir de grandes cantidades de información (Big Data), se automatice tanto el etiquetado estático como la caracterización de las conductas dinámicas del dominio.

Para cumplir con este objetivo se ha preparado un sistema que cumple con los siguientes requisitos:

- Un Sistema Operativo diseñado para servidores, CentOS, capaz de integrar las últimas herramientas de Apache Hadoop para la manipulación y almacenamiento de grandes volúmenes de datos.
- La instalación del ecosistema de herramientas Apache Hadoop distribuyendo la carga de manera óptima entre los servidores de la empresa.
- La automatización, en la medida de lo posible, de la generación de las queries de dominio que crearán las tablas transaccional y contextual que se utilizarán para alimentar la máquina de estados.

Desde el punto de vista del aprendizaje se han obtenido amplios conocimientos sobre la arquitectura que compone Apache Hadoop y sus diferentes componentes. Se ha aprendido:

- Cómo modificar un Sistema Operativo para evitar problemas de conexiones y puertos durante una instalación.
- La estructura que conforma Hadoop y sus bases.
- La utilización del entorno web de Apache Ambari y los beneficios que aporta al ecosistema Hadoop.
- Cómo utilizar HDFS y Spark para sacarle el máximo partido a Big Data y poder almacenar y ejecutar instrucciones de manera distribuida.
- Scala, como herramienta de manipulación de datos mediante Spark y en el notebook Apache Zeppelin

6.2 Trabajo futuro

El trabajo futuro pasará por completar el proyecto actual con un nuevo Trabajo de Fin de Grado titulado “Inferencia de máquinas de estado desde Big Data para el etiquetado estático y caracterización de conductas dinámicas: generador del modelo”.

Este segundo TFG parte de la base creada por el actual Trabajo de Fin de Grado para implementar un algoritmo de inferencia de máquinas de estado usando como entrada las tablas generadas por el generador de queries de dominio.

De esta forma se logrará completar el proyecto global de Gain Dynamics y con él, se podrán realizar perfiles de clientes y conductas de agentes mucho más precisos que los modelos actuales.

Referencias

[1] Asignatura “Búsqueda y Minería de Datos”, Tema 0-presentacion

[2] Big Data, gran potencial y prioridad de negocio: http://www.cisco.com/c/es_es/about/press-2013/2013-04-02-big-data-gran-potencial-y-prioridad-de-negocio.html Fecha de consulta: 08/07/2017.

[3] PoweredBy Apache Hadoop: https://wiki.apache.org/hadoop/PoweredBy Fecha de consulta: 10/07/2017

[4] Apache Hadoop: https: http://hadoop.apache.org/. Fecha de consulta: 10/07/2017

[5] Apache Spark: https://databricks.com/spark/about. Fecha de consulta: 20/07/2017

[6] Holden Karau. “Making Interactive Big Data applications Fast and Easy”. Databricks stanford.edu.

[7] Ronald L. Rivest & Robert E. Schapire. “Inference of Finite Automata Using Homing Sequences”, MIT Laboratory for Computer Science Cambridge, MA 02139

[8] Josh Bongard & Hod Lipson. “Active Coevolutionary Learning of Deterministic Finite Automata”, Journal of Machine Learning Research 6 (2005) 1651–1678, Published 10/05.

[9] Diferencias entre Hive y Hbase: https://www.xplenty.com/blog/2014/05/hive-vs-hbase/ Fecha de consulta: 16/07/2017.

[10] Hortonworks, Inc. “Apache Ambari User Guide”, 28 Nov 2016.

[11] Martin Odersky. “The Scala Language Specification Version 2.9”, Journal of programming methods laboratory epfl switzerland, Published June 11, 2014.

Glosario

API	Application Programming Interface
DMP	Data Management Platform
ETL	Extract, Transform and Load
GRUB	GRand Unified Bootloader
YUM	Yellow dog Updater Modified
THP	Transparent Huge Pages
SSH	Secure SHell
NTP	Network Time Protocol
JDK	Java Development Kit

Anexos

A Manual de configuración de CentOS 6.8

Internet
vi /etc/sysconfig/network-scripts/ifcfg-em1
cambiar
NM_CONTROLLED="no"
ONBOOT="yes"

SSH
yum -y install openssh-server openssh-clients
chkconfig sshd on
service sshd start

Instalar nano
yum install nano

Instalar wget
yum install wget
yum update

Instalar NTP
yum install ntp ntpdate ntp-doc
chkconfig ntpd on
ntpdate 0.es.pool.ntp.org

nano /etc/ntp.conf
cambiar
server 0.centos.pool.ntp.org iburst
server 1.centos.pool.ntp.org iburst
server 2.centos.pool.ntp.org iburst
server 3.centos.pool.ntp.org iburst
por
server 0.es.pool.ntp.org iburst
server 1.europe.pool.ntp.org iburst
server 2.europe.pool.ntp.org iburst
server 3.europe.pool.ntp.org iburst

/etc/init.d/ntpd start

Comprobar con el comando:
watch ntpq -cpe -cas

Conectar los dos hosts
nano /etc/hosts
añadir
127.0.1.1 gdcluster1 (en el gdcluster1)
127.0.1.1 gdcluster2 (en el gdcluster2)
192.168.1.*** gdcluster2 en el gdcluster1
192.168.1.*** gdcluster1 en el gdcluster2

/etc/init.d/network restart

Deshabilitar SELinux
En cada host:
nano /etc/selinux/config
cambiar
SELINUX=disabled

Deshabilitar iptables
chkconfig iptables off
/etc/init.d/iptables stop

Al terminar setup, se puede rehabilitar iptables

Deshabilitar Transparent Huge Pages
nano /etc/rc.local
añadir
#disable THP at boot time
if test -f /sys/kernel/mm/redhat_transparent_hugepage/enabled; then
echo never > /sys/kernel/mm/redhat_transparent_hugepage/enabled
fi
if test -f /sys/kernel/mm/redhat_transparent_hugepage/defrag; then
echo never > /sys/kernel/mm/redhat_transparent_hugepage/defrag
fi

reboot

cat /sys/kernel/mm/transparent_hugepage/enabled;
cat /sys/kernel/mm/transparent_hugepage/defrag;
cat /sys/kernel/mm/redhat_transparent_hugepage/enabled;
cat /sys/kernel/mm/redhat_transparent_hugepage/defrag

Instalar MySQL
yum install mysql-server
/sbin/service mysqld start
/usr/bin/mysql_secure_installation

Para asegurarse de que el servidor de Base de Datos se inicia después de un reinicio:
chkconfig mysqld on

Instalar Ambari server
wget -nv http://public-repo-1.hortonworks.com/ambari/centos6/2.x/updates/2.4.2.0/ambari.repo -O /etc/yum.repos.d/ambari.repo

yum repolist
yum install ambari-server

En Ambari Server host:
yum install mysql-connector-java
ls /usr/share/java/mysql-connector-java.jar
chmod 644 /usr/share/java/mysql-connector-java.jar
chmod 644 /usr/share/java/mysql-connector-java-5.1.17.jar
ambari-server setup --jdbc-db=mysql --jdbc-driver=/usr/share/java/mysql-connector-java.jar

En los servidores que vayan a albergar Ambari, Hive o Oozie ejecutar en la consola de MySQL:

CREATE USER 'ambari'@'%' IDENTIFIED BY '****';
GRANT ALL PRIVILEGES ON *.* TO ‘ambari'@'%';
FLUSH PRIVILEGES;
CREATE DATABASE ambari;

CREATE USER 'hive'@'%' IDENTIFIED BY '****';
GRANT ALL PRIVILEGES ON *.* TO ‘hive'@'%';
FLUSH PRIVILEGES;
CREATE DATABASE hive;

CREATE USER 'oozie'@'%' IDENTIFIED BY '****';
GRANT ALL PRIVILEGES ON *.* TO ‘oozie'@'%';
FLUSH PRIVILEGES;
CREATE DATABASE oozie;

ambari-server setup
ambari-server start

Instalar Ambari agent
yum install ambari-agent

nano /etc/ambari-agent/conf/ambari-agent.ini
cambiar
hostname=localhost
por
hostname=gdcluster1

ambari-agent start

Crear carpeta para HIVE
sudo su - hdfs
hdfs dfs -mkdir /user/admin
hdfs dfs -chown root:hdfs /user/admin
exit

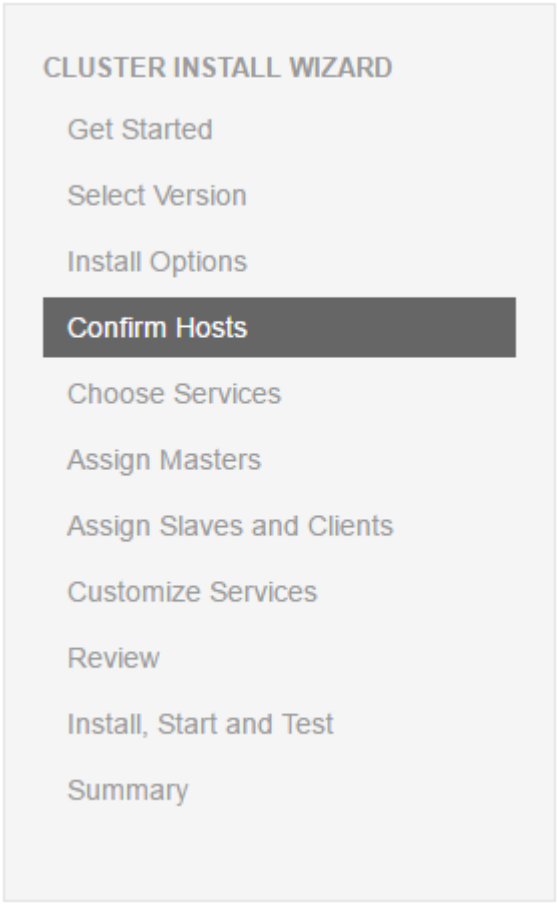
Añadir host al equipo, para Zeppelin

C:\Windows\System32\drivers\etc
Por problemas de permisos, copiar el archivo hosts al escritorio y abrir con notepad++ añadir la siguiente línea:
192.168.1.*** gdcluster1

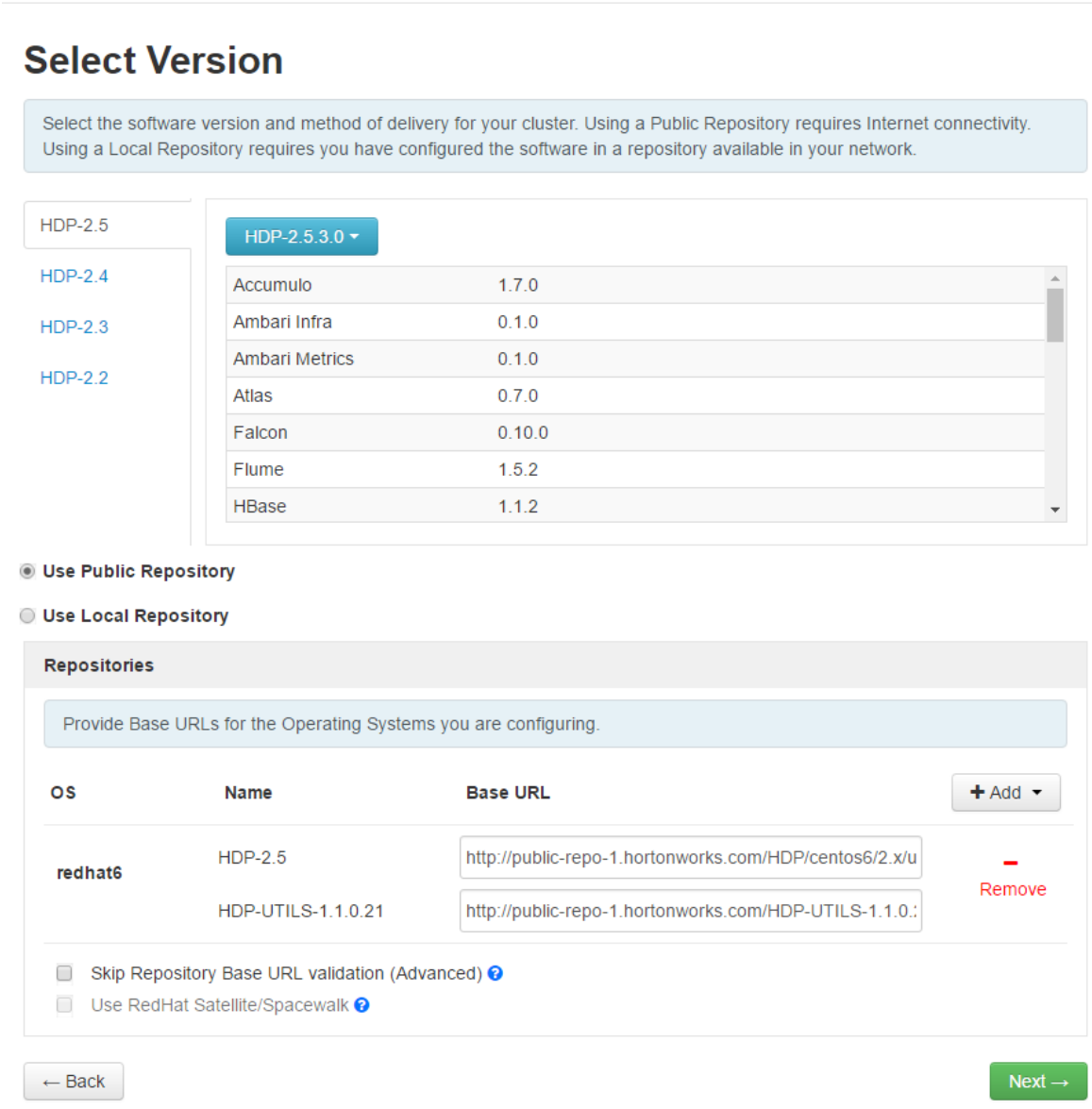
copiar el archivo hosts a C:\Windows\System32\drivers\etc

Dar permisos a Ambari para HDFS
su - hdfs
hadoop fs -mkdir /user/admin
hadoop fs -chown admin:hadoop /user/admin

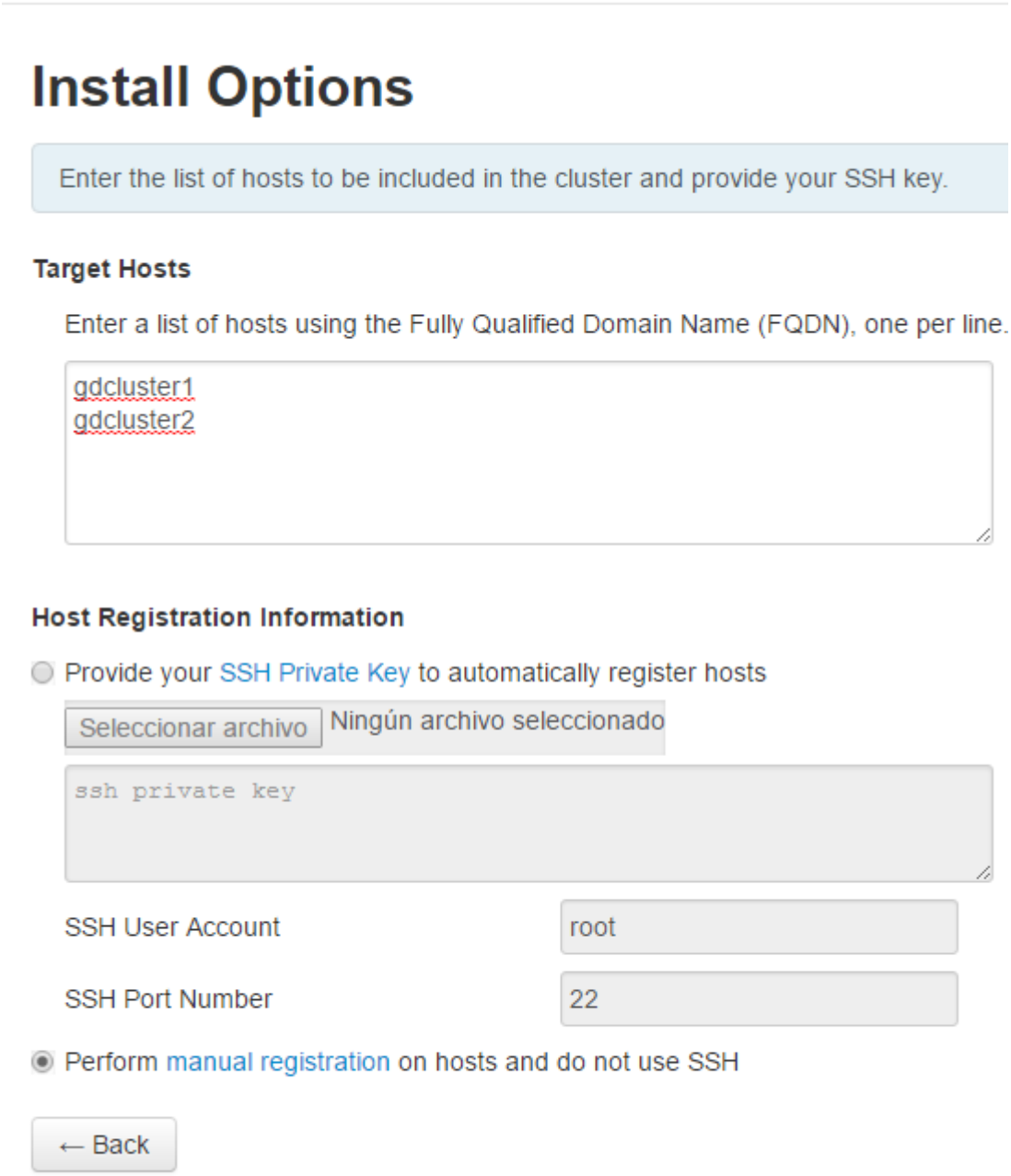
B Manual de instalación y configuración del cluster



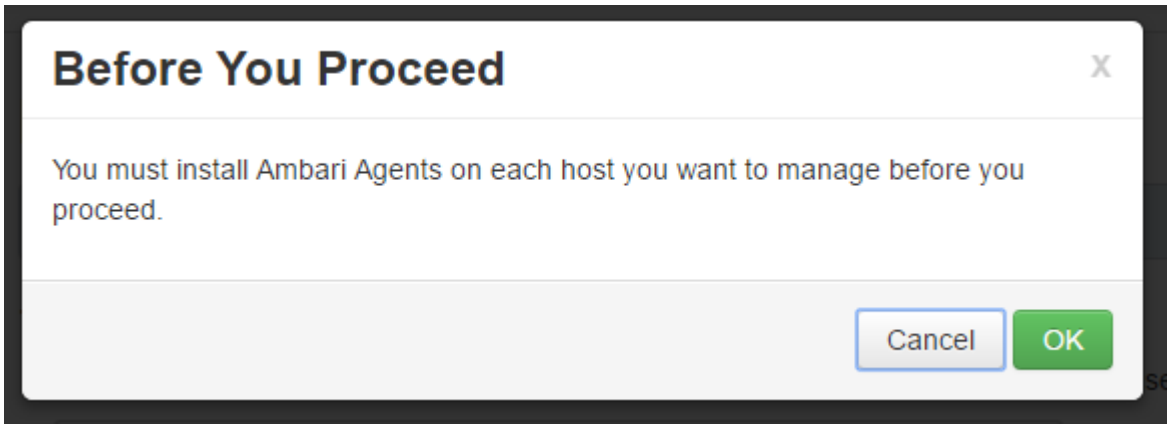
Pasos del asistente de instalación.



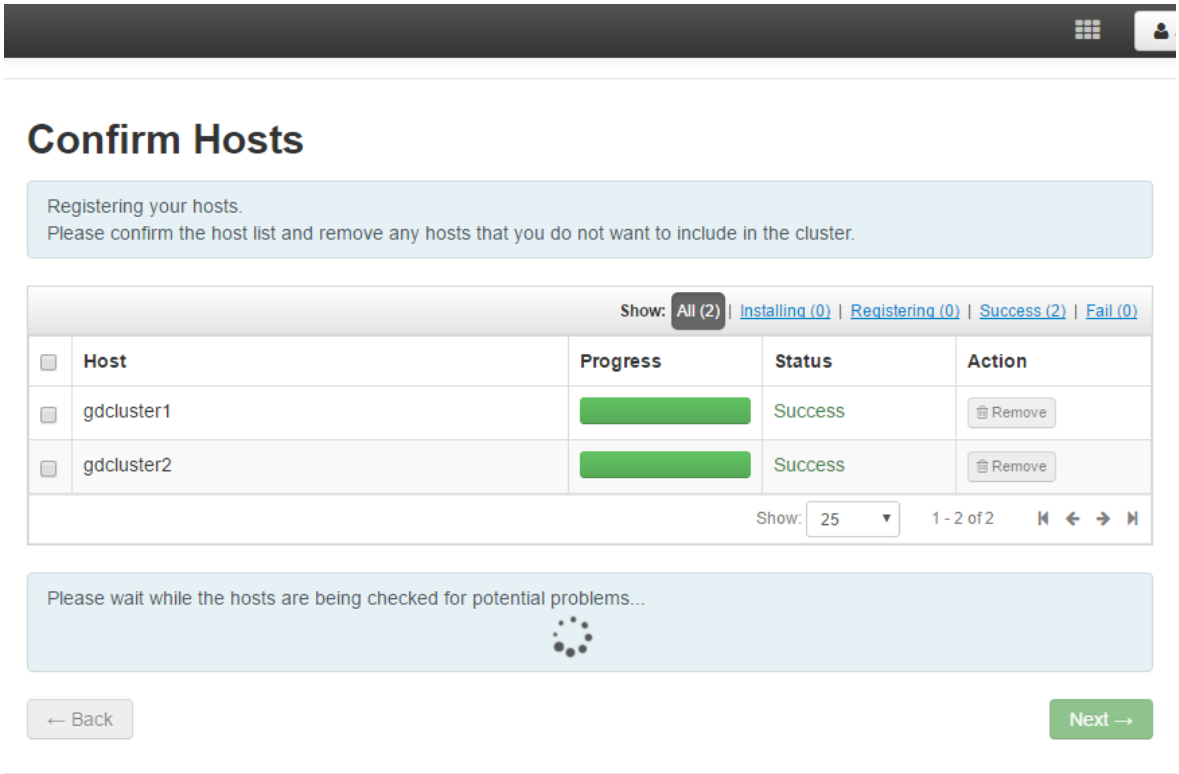
En el primer paso se selecciona la versión requerida y el repositorio del Sistema Operativo.



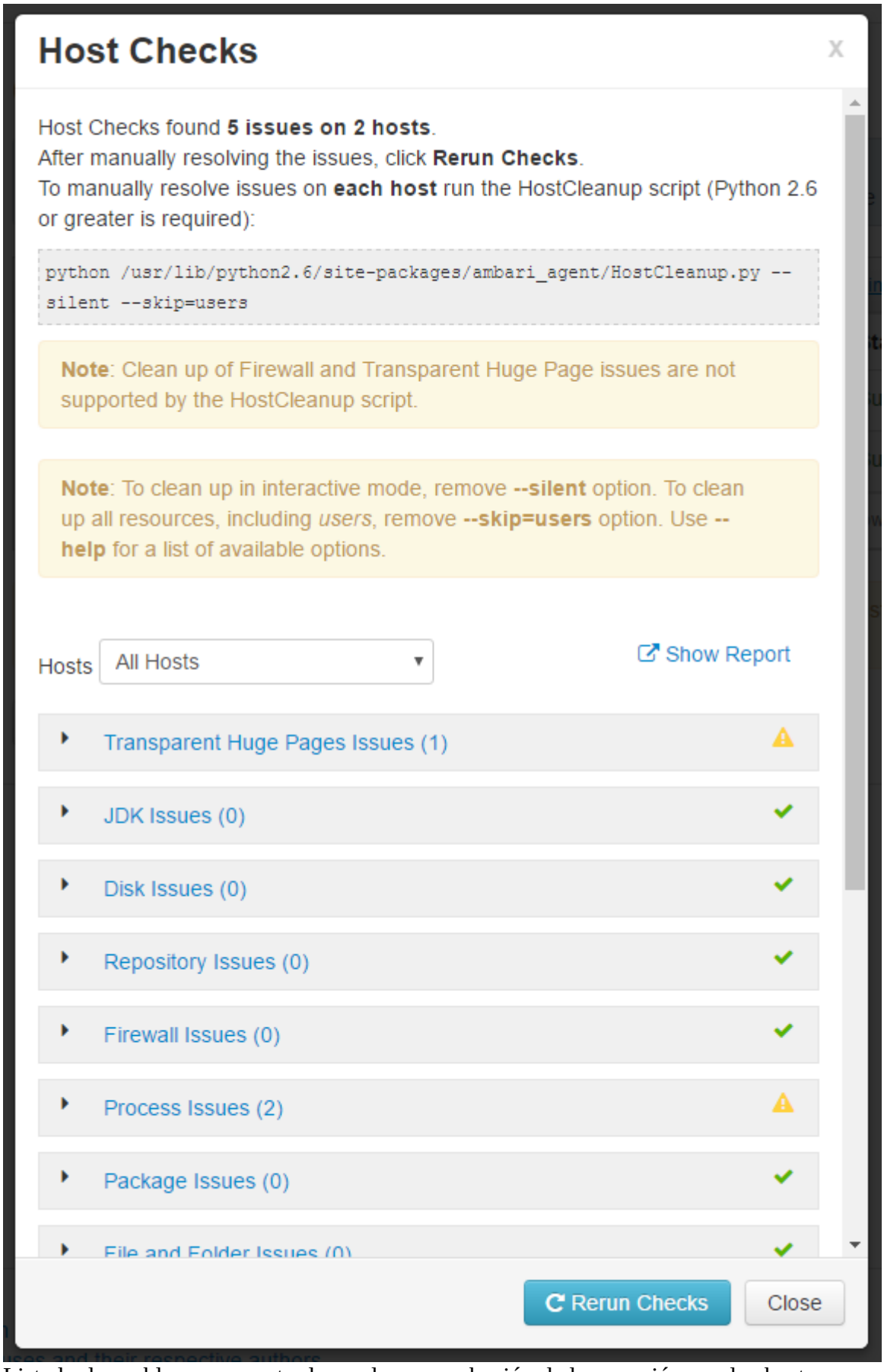
En el segundo paso se introducen los hosts del cluster y se selecciona la instalación manual de los Ambari agents.



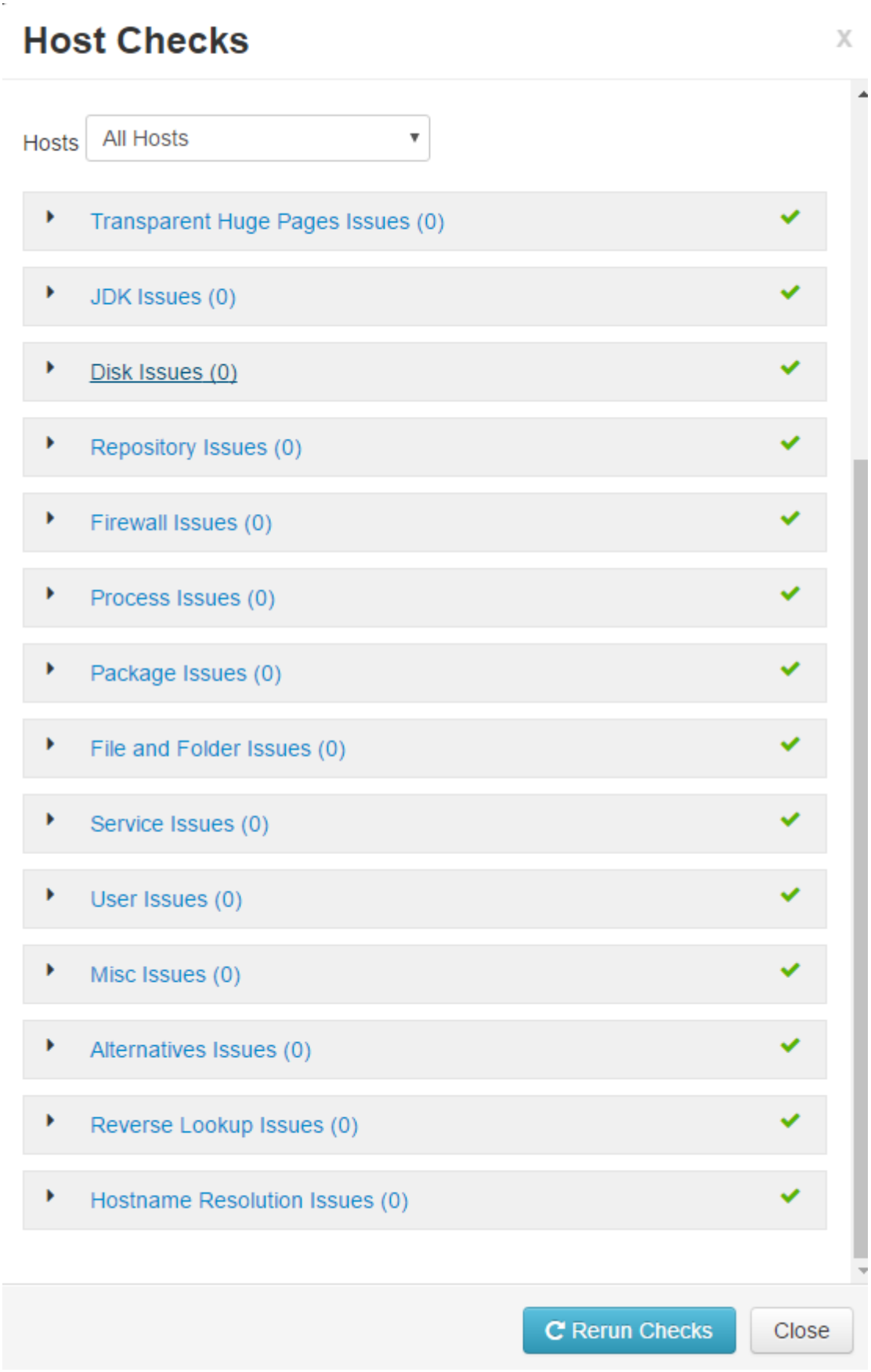
Iniciamos los agents y pulsamos OK.



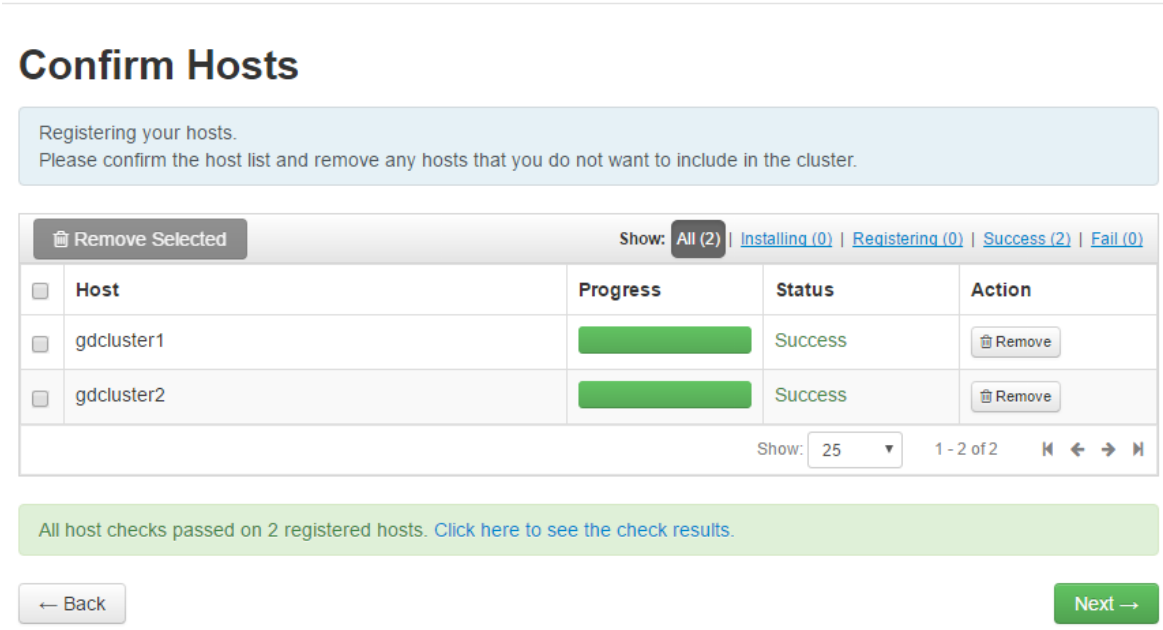
En el tercer paso se comprueba la conexión con los hosts.



Listado de problemas encontrados en la comprobación de la conexión con los hosts.



Listado una vez solucionados los problemas.

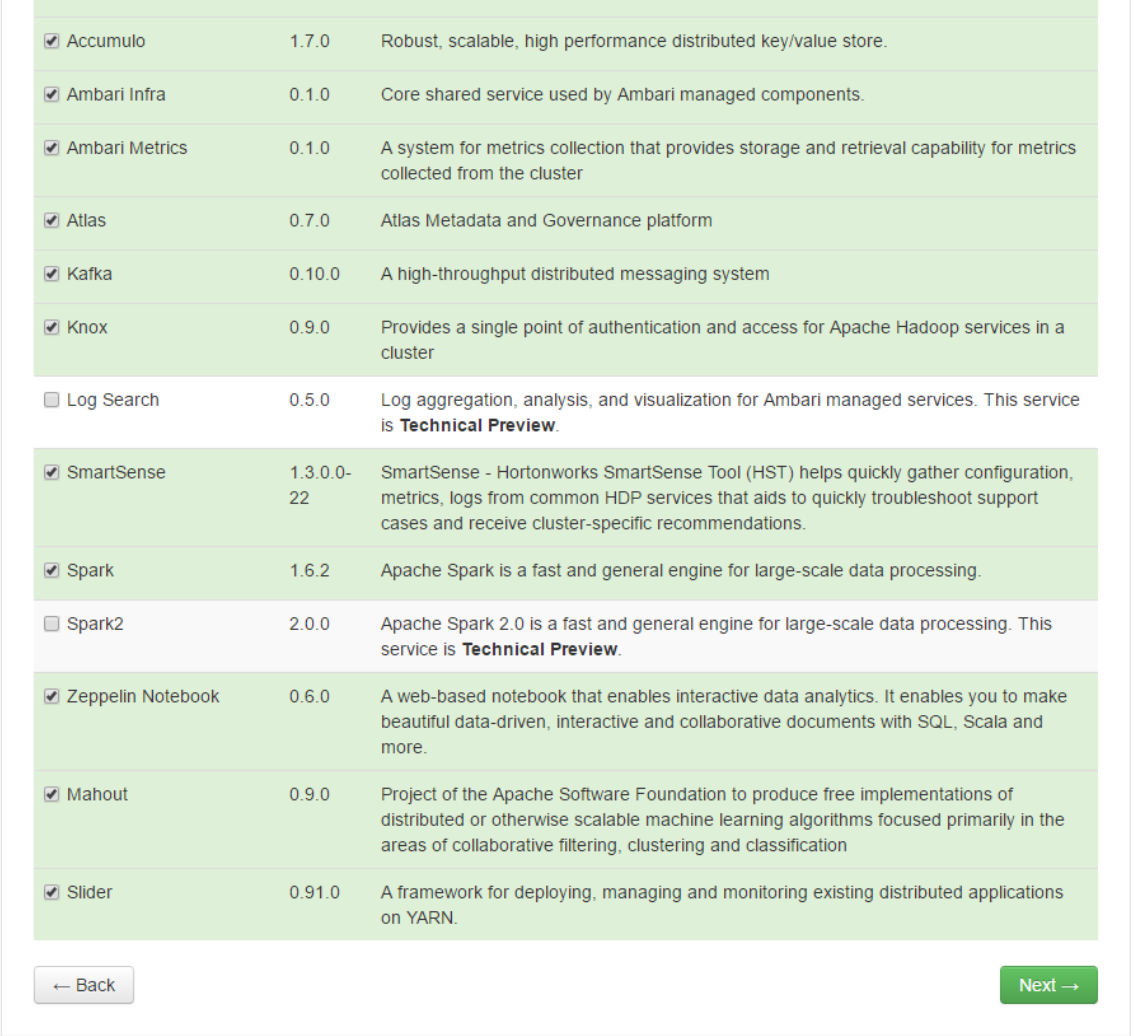


Comprobación exitosa de la conexión con los hosts tras la solución de los problemas encontrados.

Choose Services

Choose which services you want to install on your cluster.

Service	Version	Description
HDFS	2.7.3	Apache Hadoop Distributed File System
YARN + MapReduce2	2.7.3	Apache Hadoop NextGen MapReduce (YARN)
Tez	0.7.0	Tez is the next generation Hadoop Query Processing framework written on top of YARN.
Hive	1.2.1000	Data warehouse system for ad-hoc queries & analysis of large datasets and table & storage management service
HBase	1.1.2	A Non-relational distributed database, plus Phoenix, a high performance SQL layer for low latency applications.
Pig	0.16.0	Scripting platform for analyzing large datasets
Sqoop	1.4.6	Tool for transferring bulk data between Apache Hadoop and structured data stores such as relational databases
Oozie	4.2.0	System for workflow coordination and execution of Apache Hadoop jobs. This also includes the installation of the optional Oozie Web Console which relies on and will install the ExtJS Library.
ZooKeeper	3.4.6	Centralized service which provides highly reliable distributed coordination
Falcon	0.10.0	Data management and processing platform
Storm	1.0.1	Apache Hadoop Stream processing framework
Flume	1.5.2	A distributed service for collecting, aggregating, and moving large amounts of streaming data into HDFS
Accumulo	1.7.0	Robust, scalable, high performance distributed key/value store.
Ambari Infra	0.1.0	Core shared service used by Ambari managed components.
Ambari Metrics	0.1.0	A system for metrics collection that provides storage and retrieval capability for metrics collected from the cluster



☑ Accumulo	1.7.0	Robust, scalable, high performance distributed key/value store.
☑ Ambari Infra	0.1.0	Core shared service used by Ambari managed components.
☑ Ambari Metrics	0.1.0	A system for metrics collection that provides storage and retrieval capability for metrics collected from the cluster
☑ Atlas	0.7.0	Atlas Metadata and Governance platform
☑ Kafka	0.10.0	A high-throughput distributed messaging system
☑ Knox	0.9.0	Provides a single point of authentication and access for Apache Hadoop services in a cluster
☐ Log Search	0.5.0	Log aggregation, analysis, and visualization for Ambari managed services. This service is **Technical Preview**.
☑ SmartSense	1.3.0.0-22	SmartSense - Hortonworks SmartSense Tool (HST) helps quickly gather configuration, metrics, logs from common HDP services that aids to quickly troubleshoot support cases and receive cluster-specific recommendations.
☑ Spark	1.6.2	Apache Spark is a fast and general engine for large-scale data processing.
☐ Spark2	2.0.0	Apache Spark 2.0 is a fast and general engine for large-scale data processing. This service is **Technical Preview**.
☑ Zeppelin Notebook	0.6.0	A web-based notebook that enables interactive data analytics. It enables you to make beautiful data-driven, interactive and collaborative documents with SQL, Scala and more.
☑ Mahout	0.9.0	Project of the Apache Software Foundation to produce free implementations of distributed or otherwise scalable machine learning algorithms focused primarily in the areas of collaborative filtering, clustering and classification
☑ Slider	0.91.0	A framework for deploying, managing and monitoring existing distributed applications on YARN.

En el cuarto paso se seleccionan los servicios que se instalarán en el cluster.

Assign Masters

Assign master components to hosts you want to run them on.
* HiveServer2 and WebHCat Server will be hosted on the same host.

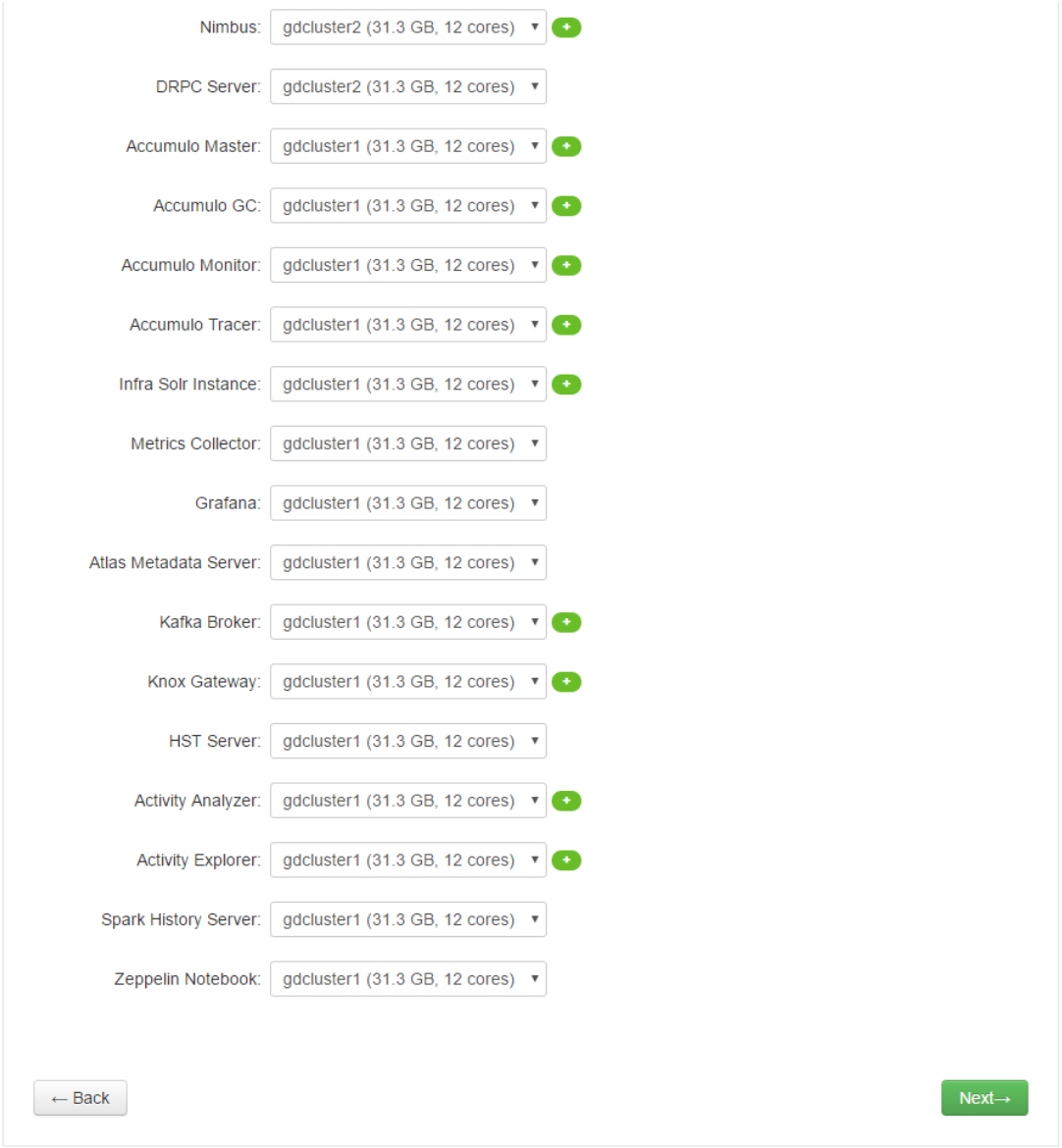
NameNode:	gdcluster1 (31.3 GB, 12 cores)
SNameNode:	gdcluster2 (31.3 GB, 12 cores)
ResourceManager:	gdcluster2 (31.3 GB, 12 cores)
App Timeline Server:	gdcluster2 (31.3 GB, 12 cores)
History Server:	gdcluster2 (31.3 GB, 12 cores)
WebHCat Server:	gdcluster2*
Hive Metastore:	gdcluster2 (31.3 GB, 12 cores)
HiveServer2:	gdcluster2 (31.3 GB, 12 cores)
HBase Master:	gdcluster1 (31.3 GB, 12 cores) +
Oozie Server:	gdcluster2 (31.3 GB, 12 cores)
ZooKeeper Server:	gdcluster1 (31.3 GB, 12 cores) -
ZooKeeper Server:	gdcluster2 (31.3 GB, 12 cores) -
Falcon Server:	gdcluster2 (31.3 GB, 12 cores)
Storm UI Server:	gdcluster2 (31.3 GB, 12 cores)
Nimbus:	gdcluster2 (31.3 GB, 12 cores) +

gdcluster1 (31.3 GB, 12 cores)

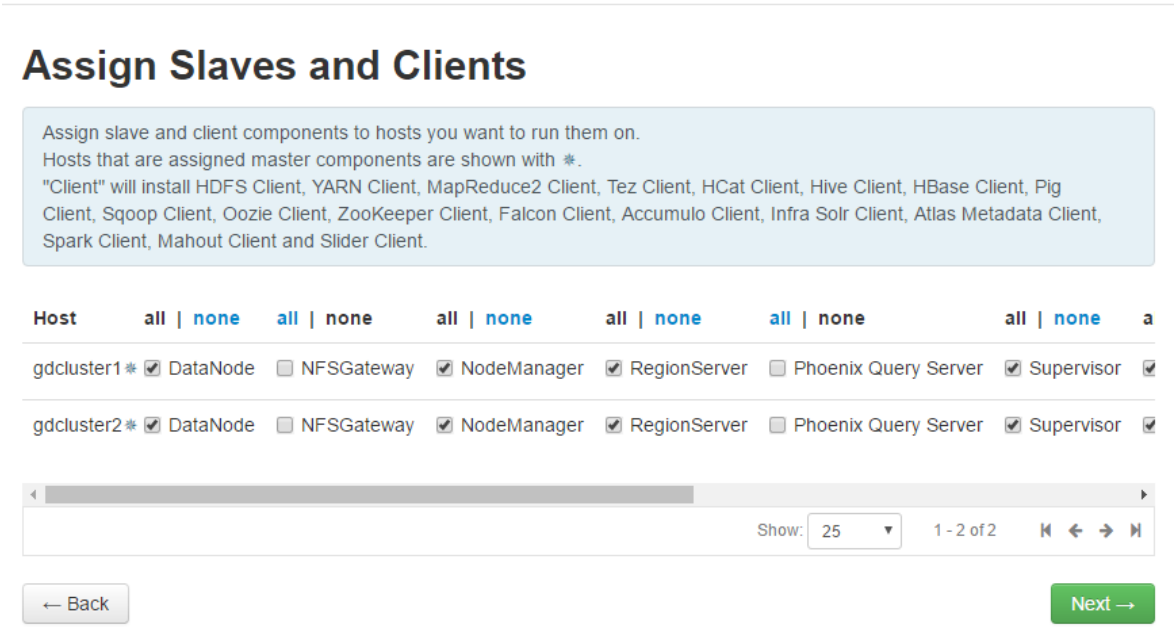
NameNode | HBase Master | ZooKeeper Server | Accumulo Master | Accumulo GC | Accumulo Monitor | Accumulo Tracer | Infra Solr Instance | Metrics Collector | Grafana | Atlas Metadata Server | Kafka Broker | Knox Gateway | HST Server | Activity Analyzer | Activity Explorer | Spark History Server | Zeppelin Notebook

gdcluster2 (31.3 GB, 12 cores)

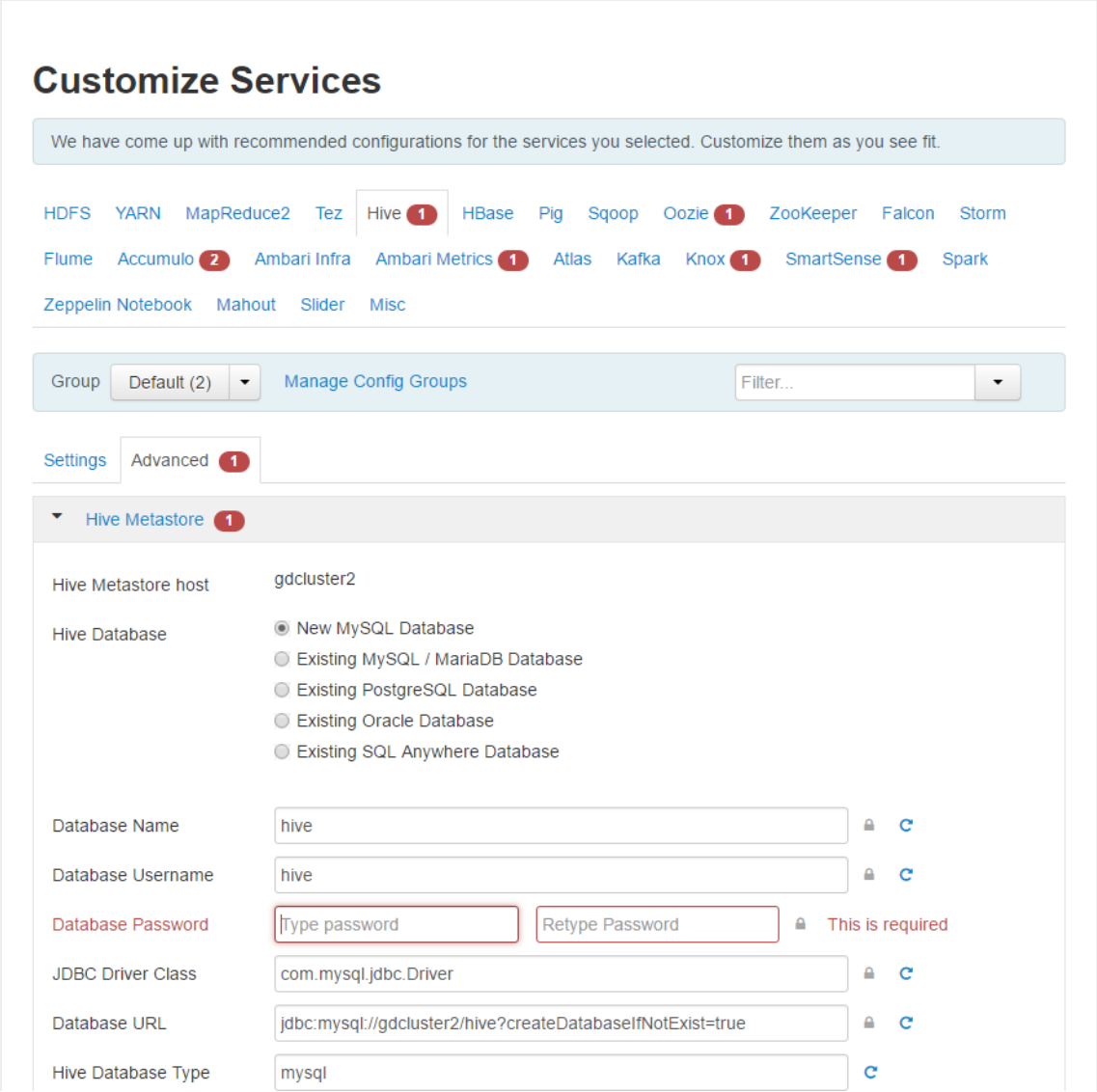
SNameNode | ResourceManager | App Timeline Server | History Server | WebHCat Server | Hive Metastore | HiveServer2 | Oozie Server | ZooKeeper Server | Falcon Server | Storm UI Server | Nimbus | DRPC Server



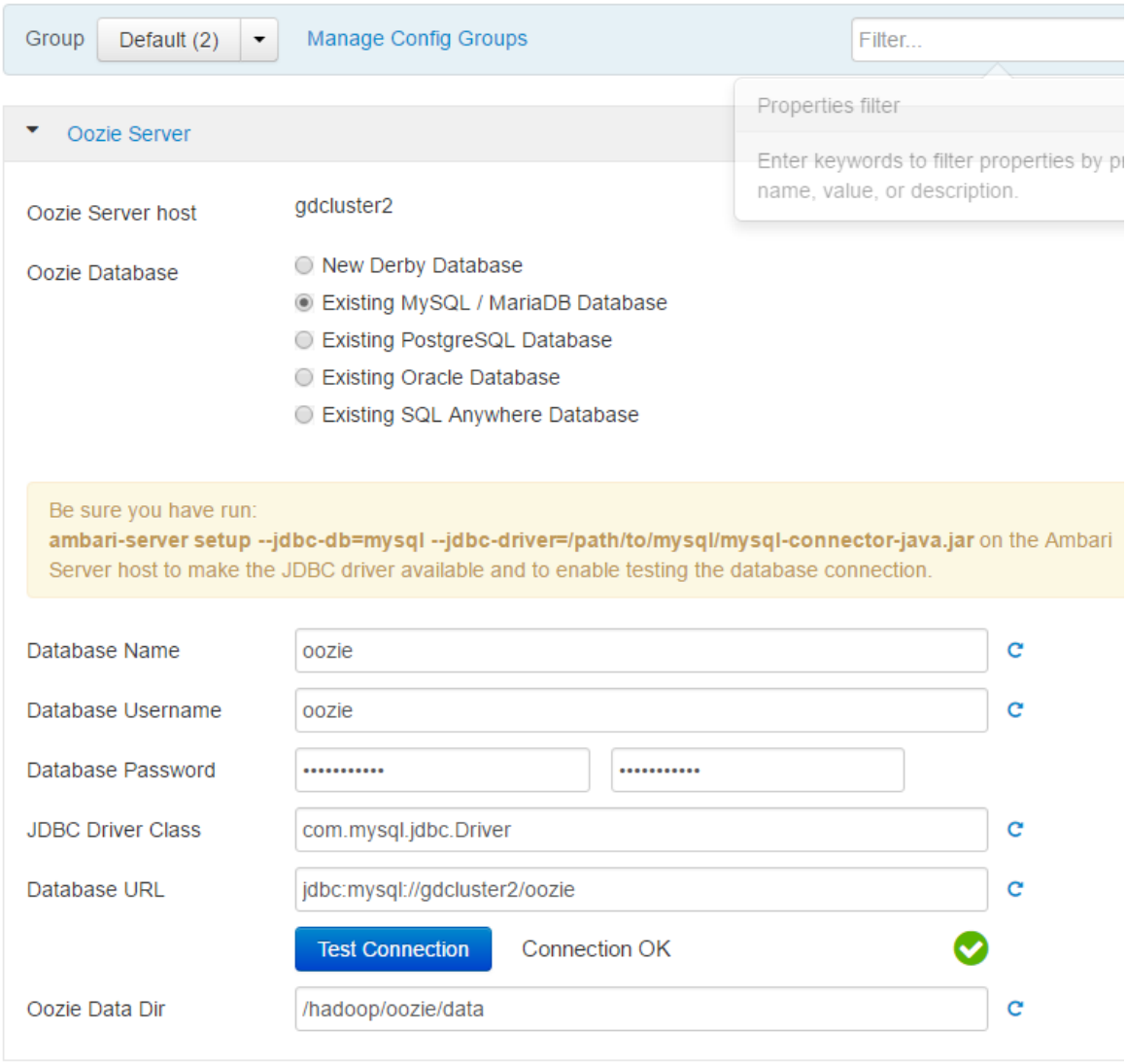
En el quinto paso se selecciona en qué hosts irán los masters de los servicios.



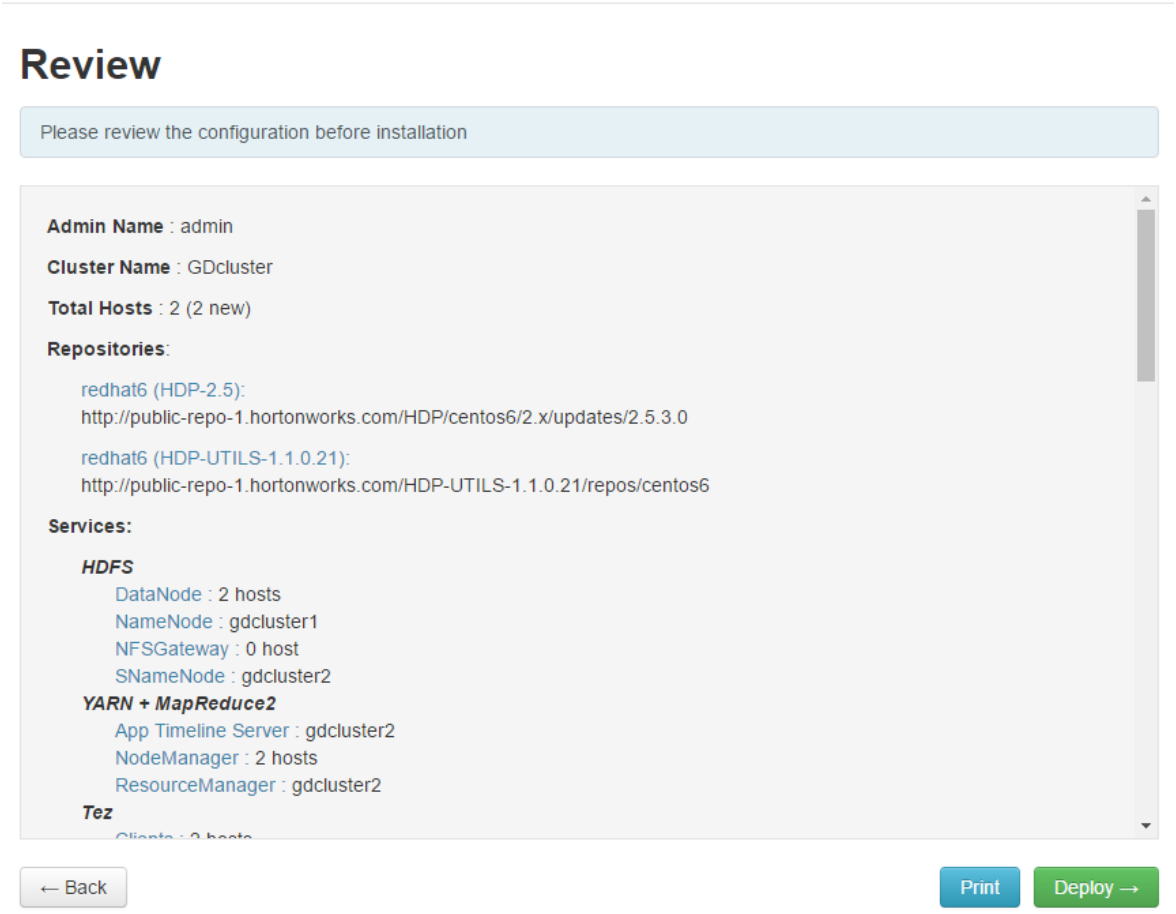
En el sexto paso se selecciona en qué hosts irán los slaves.



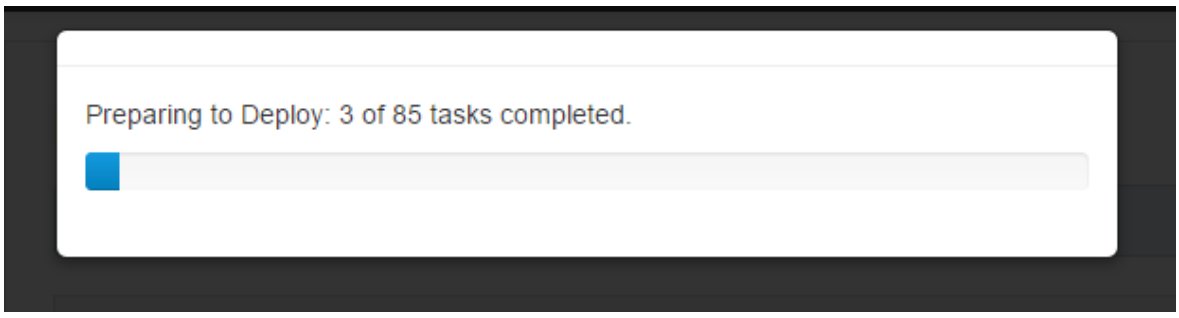
En el séptimo paso se configuran los servicios. La Base de Datos que van a usar, los límites de almacenamiento, etc.



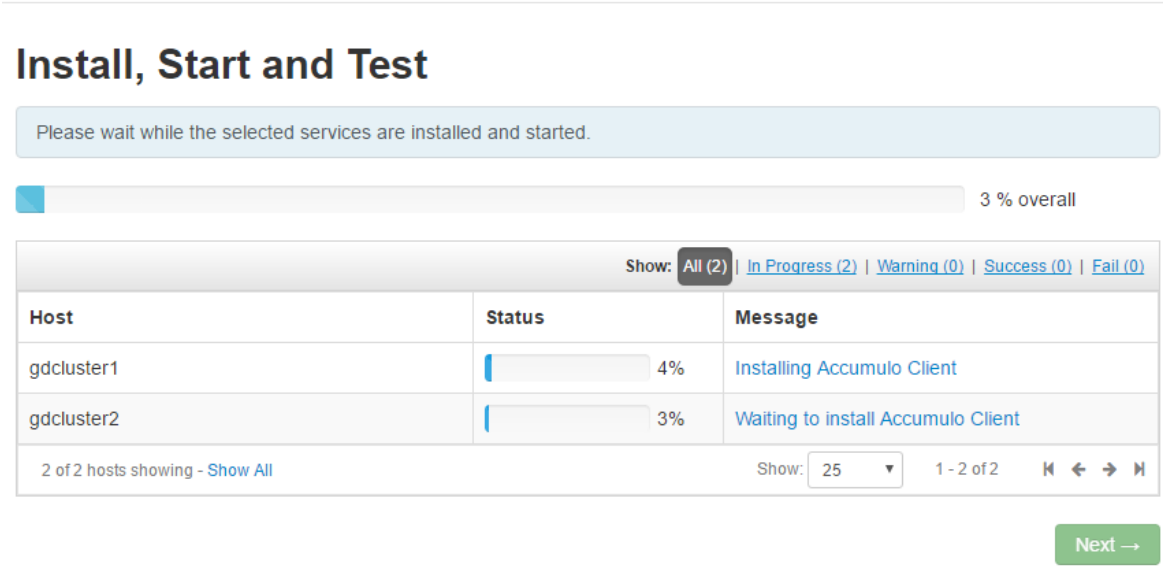
Para este proyecto seleccionamos la Base de Datos MySQL.



En el octavo paso se muestra un resumen de toda la configuración realizada.



Una vez esté todo correcto, se pulsa Deploy para empezar la instalación.



En el noveno paso se muestra el progreso de la instalación, una vez termine el cluster ya estará instalado.